Diplomarbeit

Sicheres Outsourcing mit verteilten Webservices

Konzepte der Zugriffskontrolle und ihre Umsetzung in aspektorientierter Programmierung

Prof. Dr. Claudia Eckert
Fachgebiet Sicherheit in der Informationstechnik
Fachbereich Informatik

Betreuer: Dr. Andreas U. Schmidt

Thomas Rauch   Nicolai Kuntze

17. Februar 2005



Gesetzt in LaTeX

Vorwort

Im März 2003 wurde von Frau Prof. Eckert und der Firma Guard24 eine Diplomarbeit ausgeschrieben, deren Inhalt die Entwicklung und Umsetzung eines Sicherheitskonzepts für eine webbasierte ASP-Logistiksoftware war. Der Hintergrund dieser Ausschreibung war der Wunsch der Firma Guard24 eine neue Logistiksoftware entwickeln zu lassen, die alle Geschäftsprozesse der Firma, inklusive dem Kontakt zu Endkunden, Logistikpartnern und andere Firmen, umfassen soll. Als Plattform wurde J2EE favorisiert und der Betrieb durch eine externe Serverfarm angestrebt. Es sollte untersucht werden, wie eine hohe Betriebssicherheit und ein optimaler Schutz der Daten erreicht werden können, unter Einbeziehung der hohen Anzahl an Schnittstellen zu Drittanbietern.

Von diesem Szenario ausgehend haben wir im April 2004 angefangen, ein abstrahiertes Modell zu entwerfen und eine Sicherheitsarchitektur zu entwickeln. Hierbei standen eine maximale Unabhängigkeit des Entwurfs von der zu schützenden Applikation und eine hohe Flexibilität im Mittelpunkt unserer Betrachtung.

Wir haben diese Ziele durch den Einsatz von aspektorientierter Programmierung und dem Liberty Alliance Framework erreicht.

Da diese Diplomarbeit in einem kleinen Team bearbeitet wurde, gilt es aufzuzeigen, wer für welche Teile der Arbeit verantwortlich zeichnet. Nicolai Kuntze hat die Kapitel 1, 3, 4, 6, 7 bis 7.2 und 8 bis 8.7 , Thomas Rauch die Kapitel 2, 5, 8.8 bis 8.10 und Kapitel 9 verfasst. Das Kapitel 10 und das Kapitel 7.3 sind gemeinsam entstanden.

Wir möchten uns auch an dieser Stelle bei unserem Betreuer Dr. Andreas U. Schmidt und Frau Prof. Eckert bedanken. Ohne deren Geduld, Hilfe und Anregungen wäre diese Arbeit nicht entstanden. Desweiteren bedanken wir uns beim SIT, deren Mitarbeiter uns stets so gut es ging unterstützten.

Weiter müssen wir uns für die unermüdliche Arbeit von Dominique Mähler und Elvira Rauch bedanken, die eine unendliche Anzahl von Fehlern entdeckten, die wir schon nicht mehr sahen.

Nicolai Kuntze und Thomas Rauch, 17. Februar 2005

Zusammenfassung

Identity Management erlangt eine immer größere Bedeutung, da immer mehr Firmen ihre IT-Systeme für Partner, Lieferanten oder Kunden öffnen. Die Diplomarbeit stellt einen Ansatz vor, durch den ein Zugriffskontroll- und Authentifikationssystem modular zu bestehenden webbasierten Systemen hinzugefügt werden kann. Das System muss hierfür nicht im Sourcecode vorliegen. In einer Trainingsphase wird das Modul an die zu schützende Applikation angepasst. Dies wird durch den Einsatz von aspektorientierter Programmierung in Form des AspectJ Frameworks erreicht. Für die Authentifikation kommt das Liberty Alliance Protokoll, ein zukünftiger Industriestandard, zum Einsatz, welches ein Single-Sign-On Protokoll unter Verwendung von Identity Federation implementiert. Eine mögliche Verwendung von Hardwaretoken in diesem Framework wird demonstriert und eine Weg vorgestellt, ein proaktives Verhalten in das Systems zu integrieren.

Abstract

Identity Management is becoming more and more important in business systems as they are opened for third parties including trading partners, consumers and suppliers. This thesis presents an approach securing a system without any knowledge of the system source code. The security module adds to the existing system authentication and authorisation based on aspect oriented programming and the liberty alliance framework, an upcoming industrie standard providing single sign on. In an initial training phase the module is adapted to the application which is to be secured. Moreover the use of hardware tokens and proactive computing is demonstrated. The high modularisation is achived through use of AspectJ, a programming language extension of Java.

Keywords: Identity Mangement, aspect oriented programming, single sign on, liberty alliance, pro active computing, aspectj, identity federation

Inhaltsverzeichnis

1 Einleitung

Outsourcing auf der Ebene der IT-Infrastrukturen ist inzwischen gängige Praxis bei Unternehmen und auch Behörden. Allgegenwärtiger Rationalisierungsdruck lässt inzwischen aber auch immer mehr den Wunsch aufkommen, wesentliche Teile von Geschäftsprozessen und geschäftlicher Kommunikation auf externe Service-Anbieter zu verlagern. Unter Sicherheitsaspekten ist dies jedoch stets kritisch, insbesondere wenn in komplexen Arbeitsabläufen Mitarbeitern von Partnerfirmen oder Außendienstlern begrenzter aber direkter Zugriff auf firmeneigene Datenbestände gewährt werden soll. [Meh04]

Verteilte Business-IT-Systeme sicher zu machen ist etwa im Rahmen der von der Distributed Management Task Force entwickelten Familie von Standards und mit Hilfe von Suns Toolkit für webbasiertes Unternehmensmanagement [Prob] möglich. Der Aufwand, so ein auf einer ausformulierten Sicherheitsrichtlinie beruhendes Gesamtsystem zu konzipieren und umzusetzen, ist allerdings erheblich, besonders wenn dabei bestehende Altsysteme einzubinden sind. [HSO+04] Dies ist gerade für kleine und mittlere Unternehmen nicht verträglich mit dem Streben nach Effizienz beim IT-Outsourcing.

Für viele Firmen ist es interessant geworden, ihre Software bei externen Firmen anzumieten. Die Anbieter von entsprechenden Diensten, sogenannte „Application Service Provider" (ASP), stellen in diesem Geschäftsmodell alle Dienstleistungen, die für einen reibungslosen Ablauf der Software benötigt werden, zur Verfügung. Ein solcher Vorgang wird als Outsourcing beschrieben.

Eine Weiterentwicklung dieses Konzeptes besteht darin, auch anderen Firmen Zugang zur Betriebssoftware zu gewähren und Abläufe in der Firmenkommunikation wesentlich zu beschleunigen. Das folgende Szenario soll dies verdeutlichen:

Ein Dienstleister nimmt defekte Geräte entgegen und gibt diese an den Hersteller zur Reparatur weiter. Bevor ein Gerät zum Hersteller gesendet werden kann, muss die Reparatur beantragt werden. Dieser Antrag erfolgt normalerweise über ein Callcenter oder Kommunikationsstrukturen wie Email. Da der Dienstleister einen speziellen Vertrag mit dem Hersteller besitzt, wird ihm aber erlaubt, direkt in der Software des Herstellers das Gerät zur Reparatur anzumelden. Die Software entscheidet, ob das Gerät zugelassen wird und vergibt dann die Fallnummer für die Bearbeitung. Der Dienstleister und der Hersteller haben hiervon einen Gewinn, da weniger Personal und Zeit nötig sind, um die Bearbeitung einzuleiten.

Ein Anmieten der Software bietet verschiedene Vorteile. In der Studie [SBW01] stellen A. Susarla et al. verschiedene Aspekte vor. Als besonders wichtig werden hier

- der Zugang zur besten Technologie,
- die bessere Betreuung und

- die hohe Implementierungsgeschwindigkeit des Gesamtsystems

genannt. Bei der Betrachtung der dort aufgeführten Kriterien, die zur Wahl einer ASP-Lösung führen, fällt allerdings auf, dass die Sicherheit und Verlässlichkeit eines Systems jeweils nur von ca. 20 % der Befragten als Kriterium angegeben wurde.

Eine Öffnung von internen Prozessen für Partner, Lieferanten oder auch Kunden birgt große Sicherheitsrisiken, die es in abgeschirmten Unternehmensnetzwerken in der Form nicht gibt. Hierzu wird in [BM03] festgestellt:

> concerns about security, trust, authentication, fraud and risk of loss are often cited as among the significant barriers to the growth of e-commerce

In diesem Umfeld ist die Verwaltung der Benutzer und ihrer Rechte eine große Herausforderung, wie in [Vog04] festgestellt wird. Die Verwaltung wird als User Management bezeichnet und ist Teil eines Identity Managements. Dieses Identity Management muss in ein Softwaresystem integriert werden, so dass die Methoden Identifikation, Authentifikation, Access Control, Accountability und Audit, wie sie in [SZ03] vorgestellt werden, die gestellten Anforderungen erfüllen.

Ein weiteres ASP- und Outsourcing-spezifisches Problem ist das Erzeugen der Datenbasis, auf deren Basis Authentifikation und Access Control basieren. Im Rahmen der Authentifikation muss sichergestellt werden, dass die Benutzerdaten unverändert sind und auch nicht unbefugt lesend zugegriffen werden können. Bei den Daten, die der Zugriffskontrolle zur Basis dienen, gelten diese Punkte auch. Da bei der Zugriffskontrolle benutzer- und rollenspezifische Rechte vergeben werden, ist zu hinterfragen, wie diese Rechte erzeugt werden und von wem. Bei einer fertig konfigurierten Software muss dem Hersteller vertraut werden, dass die vorgegebenen Rechte genau den gestellten Anforderungen entsprechen.

Diese Arbeit zeigt, dass es möglich ist ein generisches Sicherheitsmodul zu entwerfen, welches in eine bestehende Applikation integriert werden kann und die Eigenschaften der Authentifikation und Autorisation hinzufügt. Die Eigenschaften des Moduls werden in den folgenden drei Abschnitten definiert.

Identity Management Das Identity Management fasst alle Prozesse und Infrastrukturen zusammen, welche für eine Anlage, Verwaltung und Benutzung von „digitalen Identitäten” unter Beachtung einer Policy benötigt werden.

Neben dem bereits erwähnten User Management müssen noch andere Infrastrukturen bereitgestellt werden. Diese sind, wie in [Vog04] und [Lew03] gezeigt wird,

- der **Directory Service**, also eine Datenbank, in der die digitalen Identitäten und deren Rechte verwaltet und gespeichert sind,

- das Bereitstellen von **Authentisierungsmechanismen**, mit deren Hilfe die Identität einer Person überprüft werden kann,
- das **Access Management** zur Kontrolle, ob ein authentifizierter Benutzer das Recht hat, auf eine Ressource zuzugreifen und
- das **User Management**, durch das die Benutzerinformationen angelegt und verwaltet werden.

Neben der Definition des Identity Management über die benötigte Infrastruktur kann versucht werden, dies über verschiedene Sichtweisen zu tun. Hier werden in [Vog04] von M. Vogel folgende erwähnt:

User-/Administratorsicht Dies ist die Unterscheidung zwischen Endanwender und Administrator. Der Endanwender ist an einem einfachen Zugang interessiert. Der Administrator legt auf eine schnelle und einfache Verwaltung der Benutzer und deren Rechte Wert.

Insider-/Outsidersicht Ein Insider ist ein Mitarbeiter, für den korrekte Identitätsdaten vorliegen müssen. Ein Outsider stellt den Kunden oder Partner einer Firma dar. Deren Identitätsdaten unterliegen oft nicht so strengen Kriterien, da diese häufig durch Selbstregistrierungen der Kunden erlangt werden.

Prozesssicht Es wird zwischen dem User Access und dem User Lifecycle Management unterschieden. Der UserAccess behandelt die Autorisation, Authentifikation und Session Management. Im User Lifecycle Management werden die administrativen Tätigkeiten die sich auf Userdaten beziehen erfasst.

Weiter wird in [Vog04] auf die Folgen des Fehlens eines durchgängigen Identity Managements hingewiesen. Als Hauptrisiken werden

- Produktivitätsverlust,
- Sicherheitsrisiken und
- Nichterfüllung gesetzlicher Auflagen oder Vorschriften

genannt. Somit ist das Ziel die Einführung von unternehmensweiten Identity Mangement Lösungen. Diese können als Basis für die Einführung verteilter Administrationsstrukturen dienen, wie sie durch business-orientierte Modelle gefordert werden. Ein solches Modell wird durch das Beispiel am Anfang des Kapitels vorgestellt. Das anfangs erwähnte Beispiel erfordert, dass der Dienstleister ein Identity Management besitzt und dort seine Mitarbeiter verwaltet. Der Hersteller vertraut nun auf die Korrektheit der Benutzeridentifikation des Dienstleisters und gewährt den Mitarbeitern Zugang zu seiner Software.

Dezentrale Verwaltung und Workflows Ein Workflow ist eine Sequenz von Aufgaben, die zum Erreichen eines Ziels notwendig sind. Zwischen den Aufgaben bestehen Abhängigkeiten, die erfüllt sein müssen, bevor die nächste Aufgabe ausgeführt werden kann.

Aufgrund der weiter oben bereits beschriebenen Wünsche der Industrie nach einer stärkeren Verzahnung der Arbeitsabläufe, auch zwischen verschiedenen Firmen, erlangen Workflowsysteme eine immer stärkere Bedeutung in der Automatisierung von unternehmensübergreifenden Interaktionen. Der Begriff der Workflowsysteme beschreibt Software in Unternehmen, die den Ablauf von Vorgängen steuern (vgl. [Mel05]).

In [ACM01] stellen V. Atluri et al. über solche Systeme fest:

> Systems are inherently distributed, heterogeneous and autonomous in nature, and therefore do not lend themselves to centralized control.

Auch stellen sie fest, dass Skalierbarkeit eine wichtige Eigenschaft von Systemen ist, da viele konkurrierende bzw. verschiedene Instanzen des gleichen Workflows zur selben Zeit ausgeführt werden sollen. Daher können zentralisierte Workflow Management Systeme zu einer starken Geschwindigkeitsbremse werden (vgl. [AAA+95, DKM+97, MWW+98]).

Es muss ein Weg gefunden werden, wie die Entscheidungen über die Zulässigkeit eines Schrittes in einem Workflow möglichst dezentral getroffen werden kann. Es ist aber gleichzeitig erwünscht, dass auch andere Instanzen in Erfahrung bringen können, in welchem Zustand ein anderer Benutzer gerade ist. Dies ist für z.B. das Chinese-Wall Modell (siehe [BN89]) von Bedeutung, da ein Benutzer in einem webbasierten System durchaus mehrfach angemeldet und diese Mehrfachanmeldung auch erlaubt sein kann.

Modularität Bei der Entwicklung der Sicherheit muss gewährleistet werden, dass die gestellten Anforderungen an die Stabilität und Zuverläsigkeit eingehalten werden. Da diese Forderungen nur durch intensive Tests der Sicherheit erreicht werden können ist ein Ziel, die Sicherheit möglichst in einem Modul zusammenzufassen. Hierbei kommen auch Kostenerwägungen zum Tragen, da ein einmal entwickeltes Modul potenziell auch in anderen Applikationen wiederverwendet werden kann. Optimal ist ein Sicherheitsmodul, wenn es komplett getrennt von der Software entwickelt und an die Software durch eine Konfigurationsmöglichkeit angepasst werden kann, um eine größtmögliche Wiederverwendbarkeit zu erreichen.

Die in dieser Diplomarbeit vorgestellte Lösung wird in den weiteren Kapiteln dargestellt. Einen Überblick gibt folgende kurze Zusammenfassung der Kapitel.

Aufbau der Diplomarbeit Die Diplomarbeit teilt sich in zwei Abschnitte ein. In den Kapiteln 1 bis 6 werden die Grundlagen erläutert auf deren Basis ein Sicherheitsmodell entwickelt wird. Dieses Modell wird in den Kapiteln 7 bis 10 entwickelt. Im Folgenden werden die Kapitel dieser Arbeit vorgestellt.

Im Kapitel 2 wird ein Überblick gegeben über verteilte Authentifikationstechniken im Internetbereich. In diesem Rahmen werden verschiedene Single-Sign-On Protokolle betrachtet.

Kapitel 3 beschäftigt sich mit der verwendeten Programmiersprache Java und einer Erweiterung des Sprachkonzepts, mit deren Hilfe eine stärkere Modularisierung ermöglicht werden soll.

Danach wird in Kapitel 4 der zentrale Begriff des Referenzmonitors eingeführt und verschiedene Implementationsvarianten vorgestellt und der Einsatz einer rollenbasierenden Zugriffskontrolle motiviert.

Es folgt in Kapitel 5 eine Vorstellung von verschiedenen Authentifikationsmechanismen, die im Bereich des Internet verwendbar sind.

In Kapitel 6 werden die Voraussetzungen für einen sicheren Betrieb eines Dienstes im Internet erörtert. Außerdem wird betrachtet, wie die durch unterschiedliche Maßnahmen erreichte Sicherheit beurteilt und verglichen werden kann.

Das Kapitel 7 beinhaltet die Beschreibung des erarbeiteten Konzepts. Dort erfolgt auch eine Sicherheitsbetrachtung des vorgestellten Konzepts.

Kapitel 8 stellt ein generisches Sicherheitsmodul vor, welches das in Kapitel 7 vorgestellte Konzept realisiert. Es ist in der Lage, einem ungeschützten Programm Authentifikation und Autorisation hinzuzufügen, ohne das Programm an sich zu verändern und ohne dass der Sourcecode des Programms zur Verfügung steht.

Die Verwendung des Sicherheitsmoduls wird in Kapitel 9 demonstriert. Es wird ein vollständiger Lebenszyklus der Training und Einsatz beinhaltet beschrieben.

In Kapitel 10 werden weitere Möglichkeiten aufgezeigt, die keinen Eingang in die Implementierung gefunden haben und ein Fazit gezogen.

2 Single-Sign-On

Die heutige Geschäftswelt im Internet ist geprägt von webbasierten Diensten. Es werden über Internetseiten Dienste bzw. Informationen angeboten, die der Benutzer per Webbrowser nutzen kann. Eine Authentifikation wird in den meisten Fällen über den Vorweis von spezifischem Wissen durchgeführt. Der Benutzer gibt seine Benutzerkennung an und authentifiziert sich über die Vorlage eines vorher vereinbarten Geheimnisses, dem Passwort. Da es im Internet keinen großen Zusammenschluss von Geschäften gibt, muss ein Benutzer für verschiedene Dienste unterschiedliche Passwörter vorweisen. Dies kann zu einer Überhäufung der Benutzers mit verschiedensten Accounts und zugehörigen Passwörtern führen. Es kann z.B. passieren, dass ein Fluggast einen Flug online gebucht hat und für die Online-Reservierung eines Autos bei einer angeschlossenen Autovermietung wiederum ein neues Passwort vorweisen muss.

Die Sicherheit eines Passwortes steht und fällt mit der sicheren Verwahrung und Wahl des Passwortes. Da ein Mensch sich aber nicht unbegrenzt viele unterschiedliche kryptische Passwörter merken kann, tritt das Problem auf, wie man den Passwortwucher bändigen kann. Viele benutzen einfach das gleiche Passwort für unterschiedlichste Dienste oder schreiben sich die verschiedenen Passwörter irgendwo in der Nähe ihres Computers auf.

Eine Möglichkeit, das Problem der unterschiedlichen Accounts bzw. Passwörter zu vereinfachen, ist das Konzept des Single-Sign-On (SSO). Nach [Eck03] ist ein Single-Sign-On ein System, bei dem sich ein Benutzer in einem verteilten System nur einmal für den Zugang zu den vernetzten Rechnern authentifizieren muss. Der authentifizierte Zugang zu den anderen vernetzten Rechnern wird automatisch abgewickelt, d.h. der Benutzer muss sich nur ein Passwort merken, um alle angeschlossenen Dienste nutzen zu können. Dieses Konzept kann z.B. bei der Fluggesellschaft implementiert sein. Wenn ein Reisender einen Flug über die Webseite der Fluggesellschaft gebucht hat und zusätzlich bei dem Tochterunternehmen der Fluggesellschaft für Autovermietung ein Auto mieten will muss er nicht einen zweiten Benutzernamen und ein zweites Passwort für dessen Webseite wissen, sondern es reicht die einmalige Authentifikation bei der Fluggesellschaft. Das Tochterunternehmen vertraut der Fluggesellschaft dahingehend, dass sich der Reisende dort korrekt authentifiziert hat und übernimmt diese Benutzerdaten.

In Kapitel 2.1 wird kurz auf Identity Management eingegangen und die existierenden Single-Sign-On Protokolle beschrieben. In Kapitel 2.2 wird die Liberty Alliance vorgestellt und eine detaillierte Übersicht des Liberty-Protokollablaufs und der dazu benötigten Zusatzprotokolle und technischen Grundbausteine geliefert. Abschließend wird in Kapitel 2.6 auf das Projekt SourceID eingegangen das für die Implementierung des Protokolls in eine Applikation verwendet wird.

2.1 Identity Management in bisherigen Systemen

Um ein Single-Sign-On Konzept verwirklichen zu können, müssen die verschiedenen Identitäten eines Benutzers, die bei unterschiedlichen Organisationen gespeichert sind, verwaltet und verknüpft werden. Identity Management, wie in der Einleitung eingeführt, umfasst die Abwicklung der Authentifikation eines Individuums in einem verteilten Netz und die Verwaltung des Zugangs zu den Resourcen die dem Individuum zugeordnet sind. Im Folgenden werden die Protokolle die im Kontext von Single-Sign-On und Webapplikationen genau dieses leisten kurz erläutert und verglichen.

2.1.1 .NET Passport

Das .NET Passport Protokoll ist ein proprietäres Protokoll von Microsoft und ist als ein Service[1] von Microsoft zu verstehen. Es ist ein zentralisierter Service zur sicheren und einfachen Authentifikation von Benutzern für teilnehmende Betreiber von Webseiten. Das Hauptaugenmerk beim Passport liegt dabei auf dem „Single Sign In" Service für web-basierte Applikationen. Es soll die Zufriedenheit der Benutzer durch einfaches Sign-in und Registration erhöht werden. Der Benutzer wird davor verschont wiederholt seine Benutzerdaten einzugeben. Er speichert oft abgefragte Informationen in seinem Passport-Profil, das zentral auf den Microsoft Passport Servern gespeichert ist und von dort an Dritte nach Bedarf und Sicherheitspolitik weitergegeben wird.

Die Vorteile[DOT03] von Passport sind:

- convenient access
- enhanced user experience
- reduced costs and ease of administration

Die Tatsache, dass die Daten zentral gespeichert werden, hat aber nicht nur Vorteile. Man ist von der Sicherheitspolitik Microsofts abhängig, insbesondere was die Weitergabe von Daten an Dritte und den Zugang zu den Daten betrifft. Die Möglichkeit der anonymen Authentifikation ist bei Passport nicht gegeben, da jeder Benutzer eine global eindeutige Identität hat. Außerdem gibt es nach [KR00] einige Angriffsmöglichkeiten auf das Protokoll, die die Verwendung von Passport sehr in Frage stellen. Auch die Verwendung der höchsten Sicherheitsstufe „Strong Credential Sign-In" beseitigt nach [Eck01] nicht alle Sicherheitslücken. Zwar soll zukünftig Kerberos 5 (s.u.) in das Passport-System integriert werden und für mehr Sicherheit sorgen, aber es bleibt abzuwarten, ob die Sicherheitsmaßnahmen von Microsoft zur sicheren Verwahrung der Daten ausreichend sind. Zwei große Internetseiten, die Jobbörse Monster und das Online-Auktionshaus eBay, haben

[1] http://www.passport.com

mittlerweile ihre Zusammenarbeit mit Microsoft im Bezug auf den Passport Service wieder beendet. [Onl05].

2.1.2 Web Services Federation Language (WS-Federation)

Microsoft, IBM und Verisign arbeiten an einer Reihe von Spezifikationen für zukünftige Web Service Plattformen, „WS Security roadmap" oder „WS-*" genannt. Die Teilspezifikation Web Services Federation Language (WS-Federation) wurde im Juli 2003 veröffentlicht und hat nach [BDLD+03] als Hauptziel:

> The primary goal of this specification is to enable federation of identity, attribute, authentication, and authorization information.

Allerdings ist WS-Federation noch in der Entwicklungsphase und es gibt noch keine konkrete Implementierung. Wahrscheinlich kann erst mit der neuen Windows-Version „Longhorn" mit einem produktiven Einsatz dieser Spezifikation gerechnet werden [Rob04]. Zusätzlich gibt es bei der WS-Federation Spezifikation einige Überlappungen mit dem Identity Federation Framework (ID-FF) von Liberty Alliance (s.u.) das schon länger im Einsatz ist [LAP03a]. Auch ein wichtiger Baustein des ID-FF - die Anonymität eines Benutzers - ist im WS-Federation nur ein optionaler Bestandteil.

Zum Thema Intellectual Property Rights (IPRS) schreiben die Autoren der Spezifikation:

> the authors do not grant, either expressly or impliedly, a license to any intellectual property, including patents, they own or control

Dies bedeutet ein unkalkulierbares Risiko für Implementierer da die Kosten für eine mögliche Implementierung dieser Spezifikation in ein Produkt und daraus resultierende Kosten für den Verkauf eines solchen Produktes nicht bekannt sind.

2.1.3 Shibboleth

Die Architektur von Shibboleth sorgt für den sicheren Austausch von Autorisationsinformationen zwischen Universitätsinstitutionen.

> Shibboleth is an initiative by Internet2 member universities to develop and deploy new middleware technologies that can facilitate inter-institutional collaboration and access to digital content[Int]

Die Motivation für Shibboleth liegt hauptsächlich in der Möglichkeit, das Teilen von digitalen Resourcen unterschiedlichster Institutionen für Wissenschaftler über das Internet zu ermöglichen [LAP03b]. Es soll einem Nutzer z.B. einem Studenten möglich sein, Zugriff auf eine Ressource von einer anderen Universität

oder Institution zu bekommen. Bisher musste diesem Studenten dafür ein zweiter Account bei der anderen Institution eingerichtet werden. Mit Shibboleth soll es möglich sein, dass der Student sich an seinem Heimatinstitut anmeldet und durch die Vertrauensbeziehung zwischen den beiden Institutionen Zugriff auf die gewünschte Ressource erhält. Schibboleth will somit den Administrationsaufwand für Institutionen verringern, die untereinander digitale Resourcen teilen. Durch die starke Ausrichtung von Shibboleth auf den universitären Bereich ist es nicht direkt auf den Einsatz in der E-Commerce Geschäftswelt übertragbar. Wenn eine Firma X mit einer zweiten Firma Y eine Geschäftbeziehung pflegt heißt es noch lange nicht, dass auch automatisch ein Kunde der Firma X das Angebot der Firma Y nutzen darf. Dies ist im universitären Bereich der Fall, da hier im Normalfall allen Mitarbeitern einer Institution Zugriff auf Ressourcen gewährt werden soll.

2.1.4 Kerberos

Das Kerberos-Authentifikationssystem [SNS88] wurde im Rahmen des Athena Projektes am MIT entwickelt, mit dem Ziel Anfragen auf Netzwerkressourcen authentifizieren zu können. Mit Kerberos wird ein Single-Sign-On-Konzept realisiert. Die aktuelle Version ist 5. Ein Benutzer - Principal genannt - bekommt nur Zugriff auf eine Netzwerkresource, wie z.B. NFS oder Remote Login, wenn er eine Authentizitätsbescheinung von einem vertrauenswürdigen Kerberos-Server vorweisen kann. Der Kerberos-Server verwaltet in einer Datenbank die geheimen Schlüssel aller beteiligten Benutzer, authentifiziert die Benutzer und generiert Sitzungsschlüssel für die Kommunikation eines Principals mit einem Dienst. (vgl. [Eck03, Kapitel 10.5.4]).

2.1.5 Radius

RADIUS (Remote Authentication Dial In User Service) [RWL+00] ist ein Protokoll zur Authentifikation von Benutzern, die über einen Einwahldienst Zugang erlangen wollen. Diese Einwahldienste sind Service Provider, die dem Benutzer über ihre Einwahlknoten Zugang zum Internet gewähren. Da sich ein Benutzer über unterschiedliche Knoten einwählen kann, übergibt der Benutzer seine Authentifikationsdaten dem Einwahlknoten, der diese an einen zentralen RADIUS-Server weiterleitet. Die Authentifikationsdaten werden vom RADIUS-Server überprüft und der Einwahlknoten über das Ergebnis dieser Überprüfung informiert.

RADIUS ist eine vereinfachte Implementierung der AAA-Architektur, dies ist eine Architektur um Authentifikationen, Autorisationen und Abrechnungen von Benutzern durchführen zu können.

2.1.6 Vergleich

Ein Single-Sign-On System muß es ermöglichen eine AAA-Architektur (authentication, autorisation, accounting) zu unterstützen. Es muss die Möglichkeit bieten (vgl. [VCF+00], [Eck03, Kapitel 10.5.1])

- eine Authentifikation eines Benutzers mit beliebigen Authentifikationsverfahren durchführen zu können
- eine Zugriffskontrolle auf der Seite eines Provider zu ermöglichen
- eine abschließende Abrechnung der vom Benutzer genutzten Dienste ermöglichen.

Im Umfeld von Electronic Commerce sind weitere Daten wie z.B. Bezahlungsdaten und Versanddaten für einen Kauf sowie statistische Daten über das Benutzerverhalten wichtig. Ein weiterer Faktor ist die Tatsache, dass bei einem Großteil der Benutzer die Bereitschaft sehr klein ist, zusätzliche Software zu installieren, sei es zur einfacheren Nutzung oder aus Sicherheitsgründen [PW02].

Single-Sign-On Protokolle für Electronic Commerce müssen deshalb mehr als nur eine AAA-Architektur unterstützen können und auf der Client-Seite möglichst mit nichts außer dem Kernbetriebssystem und dem Browser zurechtkommen. In [PW02] wird ein Protokoll, das diese Anforderungen erfüllt, deshalb als ein „browser-based attribute-exchange" Protokoll bezeichnet und umfaßt unter anderem:

Browser-based Die Unterstützung möglichst aller Browser auf möglichst allen Plattformen.

Case-by-case attribute exchange Den freie Austausch von Atributen, die nicht jedesmal zwingend die Authentifikationsdaten beinhalten.

Cross trust domain Der Austausch von Informationen zwischen Parteien, die zu unterschiedlichen Domains gehören.

compatible with zero-footprint Benutzbar nur mit den Möglichkeiten eines Browsers. Das Protokoll muß funktionieren, auch wenn auf dem Client-Rechner außer dem Browser keine spezielle Software zum Austausch von Attributen vorhanden ist.

compatible with mobility Benutzbar für mobile Benutzer die verschiedene Devices wie z.B. Computer, Handhelds, Internet-Cafe Computer verwenden.

Die oben beschriebenen Single-Sign-On Protokolle basieren fast alle auf zusätzlicher Infrastruktur. Es müssen extra Server aufgebaut und teilweise extra Clientsoftware installiert sein, um das entsprechende Protokoll abwickeln zu können.

Das .Net Passport Protokoll funktioniert auf der Client-Seite mit den Boardmitteln des Browsers, benötigt aber zwingend den Passport-Server von Microsoft zur Authentifikation des Benutzers. Das WS-* Protokoll erfüllt viele der oben genannten Punkte, ist aber noch in der Entwicklungsphase und konkurriert mit bereits etablierten Protokollen. Das Kerberos-Protokoll benötigt zur Kommunikation auf allen Seiten kompatible Software und ist somit nicht mit den Boardmitteln eines Browsers verwendbar. Radius bietet zwar die Möglichkeit einen Benutzer über eine Webschnittstelle an einen Radius-Server anzumelden, allerdings ist es nicht speziell für den Einsatz in Electronic Commerce Applikationen ausgelegt. Auch verfügt es nicht über die Möglichkeit der Pseudonymisierung eines Benutzers. Das Shibboleth Protokoll ist in einer rein webbasierten Umgebung lauffähig, ist aber durch die starke Ausrichtung auf den universitären Bereich nicht für Electronic Commerce Applikationen geeignet. Ein Protokoll das schon etabliert ist und den gesamten Anforderungskatalog aufgreift und umsetzt ist das Liberty Protokoll und wird im folgenden genauer erläutert.

2.2 Liberty Alliance

Die Liberty Alliance[2] ist ein Zusammenschluss von verschiedenen Organisationen und Firmen der im September 2001 gegründet wurde. Ziel des Bündnisses ist der Aufbau eines offenen Standards für **Federated Identity Management**.

Zu einem solchen Standard gehört im Einzelnen:

- einen offenen Single-Sign-On Standard sowie
- eine „network identity/federated identy" Spezifikation zu definieren
- Geschäften zu ermöglichen, Beziehungen zu ihren Kunden selbst ohne Dritte handhaben zu können und
- Benutzern zu ermöglichen, dass Sie über ihre Privatsphäre und Sicherheit ihrer „network identity" bestimmen können

Nach [LAP03c] versteht die Liberty Alliance unter digitalen Indentitätsinformationen, bei Liberty Alliance „digital identity" genannt, die Online-Identität(en) eines Benutzers.

Hierzu gehören die drei AAAs, also Authentifikations-, Autorisations- und Abrechnungsdaten sowie persönliche Daten, personalisierte Online Einstellungen und Benutzergewohnheiten beim Einkaufen und Einkaufsvorlieben des Benutzers. Diese Daten sind in bisherigen Systemen weit verstreut über die einzelnen isolierten Accounts auf den speziellen Internetseiten.

Seine Identitätsinformationen soll der Benutzer selbst verwalten und sicher Organisationen seiner Wahl mitteilen können. Das Architekturmodell, das für die

[2] http://www.projectliberty.org

Absicherung und korrekte Verwendung dieser Daten zuständig ist, wird von der Liberty Alliance „federated network identity model" genannt. Unter „federated" ist im Kontext von Liberty dabei der Zusammenschluss von network identities gemeint, also der Zusammenschluss von unterschiedlichen Online Identitäten eines Benutzers, die zusammen seine „network identity" ergeben.

Da das Ziel einer eindeutigen network identity nicht auf Anhieb mit inhomogenen Systemen vereinbar ist, hat die Liberty Alliance mehrere Spezifikationen zu seiner schrittweisen Umsetzung veröffentlicht:

ID-FF (Liberty Identity Federation Framework, Version 1.0 von 2001, seit Ende 2003 Version 1.2) Der Fokus des ID-FF liegt auf der Verknüpfung, „federate", von Zugangsdaten und dient als Basis für ein Single-Sign-On System. Die oben genannte digital identity beinhaltet in dieser Spezifikation nur die Zugangsdaten zu einem Webdienst-Account.

ID-WSF Die Phase zwei der Liberty Alliance Aktivitäten ist die Spezifizierung des Liberty Identity Web Services Framework und wurde Anfang 2004 mit der Version 1.0 herausgebracht. ID-WSF baut auf dem ID-FF Protokoll auf und umfasst nun auch AAA-Daten sowie persönlichen Daten eines Benutzers, die frei definiert werden können. ID-WSF spezifiert ein Protokoll und Sicherheitsmaßnahmen, um diese Daten zwischen Providern austauschen zu können. Das Protokoll legt somit die Grundlage für den Austausch von umfangreichen Identitätsdaten zwischen Diensteanbietern.

ID-SIS Die dritte und letzte Phase der Liberty Alliance Spezifizierungen ist die „Liberty Identity Services Interface Specifications". Diese ist noch nicht komplett abgeschlossen und soll aufbauend auf ID-WSF die Möglichkeit von personenbezogenen Profilen, wie Kalender, Adressbücher, etc., bieten, mit denen die Grundlage für ein weites Feld von Anwendungsfällen bereitet wird.

Da das Liberty Protokoll für sehr viele Programme offen sein will, basiert es auf den Standardprotokollen der webbasierten Systeme. Dazu gehören HTTP, SSL, SOAP und SAML. Zusätzlich stehen die Funktionen des Browsers und des Web-Servers zur Verfügung. Da aber auch hier eine breite Basis an Browsern und Servern unterstützt werden soll, werden nur Möglichkeiten eingebunden, die schon in sehr vielen Browsern bzw. Servern integriert sind. Die Besonderheiten und Schwachstellen der verwendeten Subprotokolle und Funktionen werden im Folgenden genauer betrachtet.

2.3 Schlüsselkomponenten von Liberty

Nach [Proa] sind die Schlüsselrollen des Liberty ID-FF:

Principal Beim ID-FF Protokoll wird der Benutzer als Principal bezeichnet, der mit mindestens zwei Diensten in Kontakt steht.

IDP Ein möglicher Dienst ist der Identity Provider (IDP). Dieser ist ein Anbieter, der mit mehreren anderen Anbietern Geschäftsbeziehungen hat z.B. eine Fluggesellschaft, die mit verschiedenen Autovermietungsfirmen und Hotels kooperiert. Der IDP baut dabei den Circle of Trust (s.u.) letztlich über vertragliche Beziehungen auf.

SP Als zweites gibt es den Service Provider (SP). Dieser ist ein beliebiger webbasierter Diensteanbieter oder Informationsanbieter und umfasst praktisch jede Organisation im Web, vom einfachen Internetportal über Banken bis zu Behörden.

Circle of Trust Nach [Proa] ist ein Circle of Trust (Siehe Abb.1) eine Vereinigung von Service Providern und Identity Providern, die Geschäftsbeziehungen basierend auf der Liberty Architektur haben. Als Beispiel eines Circle of Trust

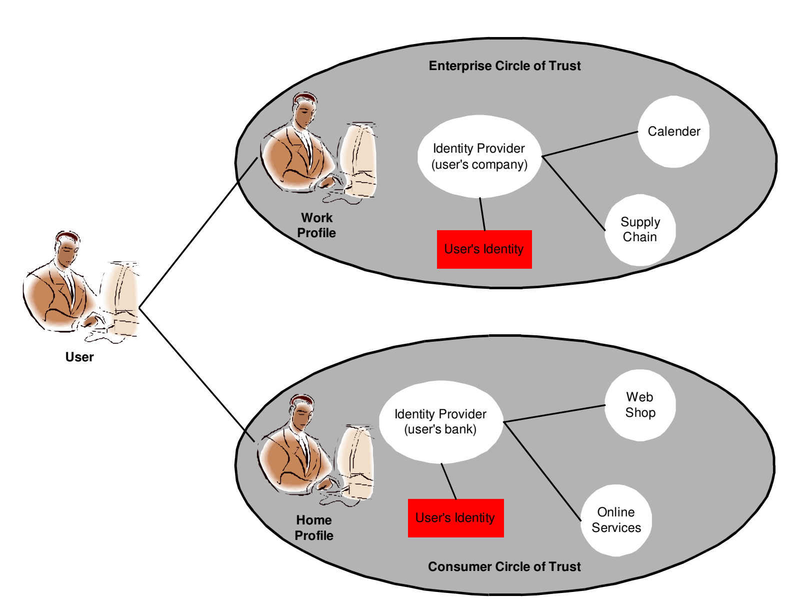


Abbildung 1: Circle of Trust [LAP03c]

dient das Szenario einer Fluggesellschaft, zu deren Service es heute gehört, dass der Fluggast zusätzlich zu der reinen Flugbuchung auch ein Hotelzimmer buchen oder ein Leihauto mieten kann. Die Fluggesellschaft hat Vertragsbeziehungen zu weiteren Firmen, die Zusatzleistungen für den Fluggast liefern und bildet entsprechend des Protokolls ein Circle of Trust. Die Fluggesellschaft ist der IDP und die angeschlossenen Firmen sind die SPs. Damit sichergestellt ist, dass nur den erlaubten Vertragspartnern Zugriff gewährt wird, werden diese bei jedem Provider

(SPs und IDPs) in eine Tabelle fest eingetragen. Es gibt dabei keine Begrenzung an wievielen Circles of Trust ein Provider teilnimmt, sodass sich transitive Beziehungen ergeben können.

Federated Digital Identities Falls ein SP und ein IDP im Sinne von Liberty eine Beziehung innerhalb eines Circle of Trust aufgebaut haben, hat der Benutzer die Möglichkeit seine beiden Accounts, beim SP und beim IDP, miteinander zu verknüpfen (federate). Er muss sich dabei auf beiden Portalen einloggen und angeben, dass er die beiden Accounts verbinden will. Falls er die beiden Accounts föderiert hat, muss er sich bei einem erneuten Login nur noch beim IDP authentifizieren wo er sich mit seinen Logindaten des IDPs anmeldet. Falls der Benutzer vom IDP akzeptiert wird, reicht dieser ihn weiter zum SP der dem IDP vertraut, dass sich der Benutzer korrekt authentifiziert hat.

Da sich sowohl SP als auch IDP merken, welchen lokalen Account sie mit dem entfernten Account föderiert haben, ist es dem SP möglich, auf den lokalen Benutzer zu schließen, sodass der Benutzer seinen Account beim SP nutzen kann ohne sich dafür beim SP ein zweites Mal authentifizieren zu müssen.

Simplified Sign-On Der IDP ist beim ID-FF Protokoll ein Single-Sign-On Server, der das Login-Portal des Circle of Trust darstellt. Dieser teilt seinen im Circle of Trust angeschlossenen SPs bei Bedarf mit, ob und wie sich der Benutzer bei ihm authentifiziert hat. Somit ist dem Benutzer ein vereinfachtes Sign-On ermöglicht. Nachdem er seine Accounts beim IDP und den SPs föderiert hat, kann er nach einer initialen Authentifikationphase beim IDP alle Dienste der föderierten SPs direkt nutzen. Falls es transitive Beziehungen über mehrere Circles of Trust gibt, funktioniert simplified Sign-On auch. Da es zur Zeit im Kontext von Webservices normal ist, sich über eine Kombination aus Benutzernamen und Passwort zu authentifizieren, liegt hierauf auch das Hauptaugenmerk des Liberty ID-FF Protokolls. Liberty limitiert sein Protokoll dabei nicht nur auf diese Methode, sondern hat das Protokoll für weitere Wege der Authentifikation offen gelassen.

Single logout Das Liberty ID-FF Protokoll bietet außerdem die Möglichkeit des single logout. Hierbei kann sich der Benutzer auf den Webseiten des gerade angezeigten Providers ausloggen und ist automatisch bei allen Providern innerhalb des Circle of Trust abgemeldet. Dies bringt den Vorteil, dass sich der Benutzer zum Beenden seiner Sitzung nicht bei allen Service Providern einzeln abmelden muss.

Pseudonym Das Liberty ID-FF Protokoll verwendet nicht, wie viele andere Ansätze, global eindeutige IDs zur Authentifikation, sondern es werden Pseudonyme für die Verknüpfung von Accounts verwenden. Zwei Accounts bei unter-

schiedlichen Providern eines Circle of Trust werden verknüpft, indem beide Seiten jeweils ein beliebiges Pseudonym für den Account generieren, dieses unter dem jeweiligen Account lokal speichern und dem Gegenüber senden. Somit muss immer nur das Pseudonym übertragen werden und es sind nur die zwei beteiligten Provider in der Lage, das Pseudonym zu ihren Accounts zuzuordnen. Dies wird für jede Verknüpfung eines Account neu generiert, sodass bei mehreren transitiven Beziehungen die Accounts eines Benutzer mit einer Kette von verschiedenen Pseudonymen verbunden sind.

Anonymität Unter Anonymität wird bei Liberty verstanden, dass ein Provider, der nur einzelne Attribute der Benutzeridentität wissen muss, auch nur diese zur Verfügung gestellt bekommt. Das könnte zum Beispiel ein Wetterdienst (Service Provider) sein, der nur die Postleitzahl zur Bestimmung des Wohnortes eines Benutzers braucht. Es wird durch diese Attributs-Sparsamkeit auf Seiten des Service Provider ein gewisser Grad an Anonymisierung erreicht bzw. eine De-Pseudonymisierung vermieden, sodass es dem Service Provier so nicht möglich ist auf die Identität eines Benutzers zu schließen. Allerdings können Service Provider und Identity Provider zusammen auf die Identität des Benutzers schließen, da das Pseudonym beiden bekannt ist und der Identity Provider die Identität des Benutzers zu diesem Pseudonym kennt.

2.4 Protokollaufbau

Nach [CK03] besteht das Liberty Protokoll aus 7 Protokollkomponenten die als Profile bezeichnet werden. Diese Profile spezifizieren die technische Implementierung der oben genannten ID-FF Schlüsselkomponenten. Das wichtigste Profil ist das **Single Sign-On and Federation** Profil und wird im Folgenden mit den verwendeten Sub-Protokollen und Techniken genauer erläutert. Eine detaillierte Beschreibung aller wichtigen Profile findet man in Liberty ID-FF Bindings and Profiles Specification [CK03].

Zur Vereinfachung wird angenommen, dass der Benutzer bei seinem Service Provider und Identity Provider föderierte Accounts hat. Die hierfür nötigen Schritte werden durch die anderen Profile des Liberty Protokoll spezifiziert. Weiterhin wird angenommen, dass sich der Benutzer noch nicht bei einem der Provider authentifiziert hat. Das Profil (siehe Abbildung 2) beginnt mit der HTTP-Anfrage eines Benutzers zur Nutzung eines Service Provider Dienstes (1). Der Service Provider muss ermitteln (2), welcher seiner eingetragenen IDPs für den Benutzer passend ist. Dies wird als das Introduction Problem bezeichnet und wird durch das **Identity Provider Introduction** Profil abgewickelt. Daraufhin schickt (3/4) der SP den Browser des Benutzers über ein Web Redirect (siehe Seite 23) zu dem Identity Provider und übergibt ihm die Rückkehradresse und eine Aufforderung zur Authentifikation des Benutzers (**AuthnRequest** - s.u.). Der Identity Provider führt die Authentifikation (5) durch und antwortet über

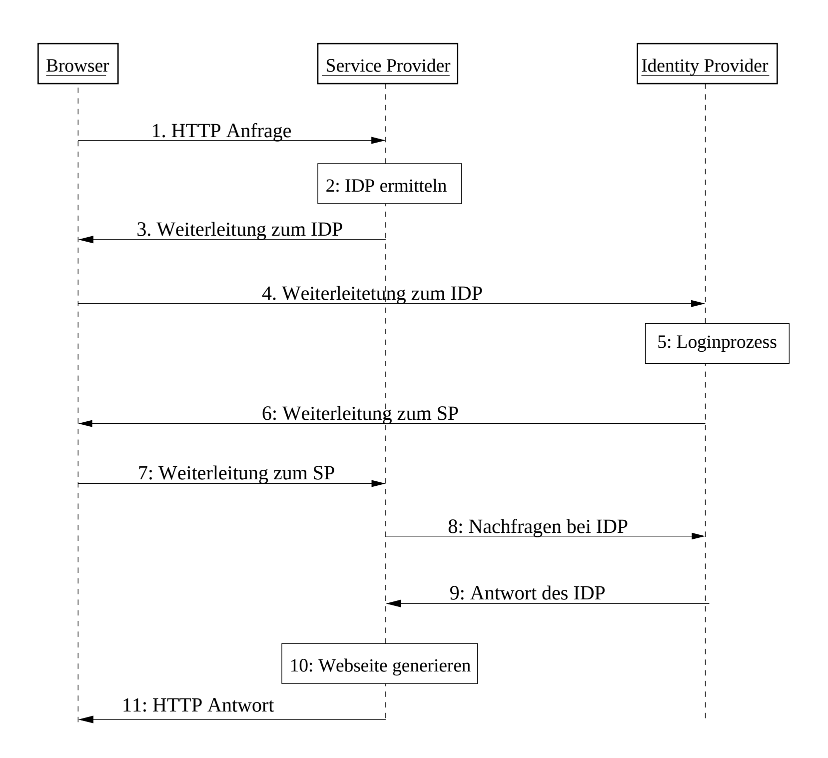


Abbildung 2: Single Sign-On and Federation

den Browser des Benutzer dem SP (6/7) mit einer **Authentication Assertion** (s.u.), **SAML Artifact** (siehe Kapitel 2.4 auf Seite 25) oder einer Fehlermeldung. Falls mit einer SAML Artifact, eine Zufallszahl, die auf eine beim IDP gespeicherte SAML Assertion deutet, geantwortet wurde, fragt (8) der SP beim IDP mit dem Artifact nach der SAML Assertion. Der IDP antwortet (9) dem SP und schickt ihm die Assertion. Wenn die Assertion richtig ausgestellt wurde, kann der SP die Webseite generieren (10) und dem Browser des Benutzers schicken (11).

Der gerade beschriebene Protokollablauf soll möglichst von einer breiten Basis an Browsern unterstützt werden, sodass das Protokoll nur einige Sub-Protokolle und Möglichkeiten verwendet, die von viele Browser unterstützt wird. Zu diesen Möglichkeiten gehören Web Redirects und die Verwendung von Cookies sowie das Verpacken von Nachrichten mit Hilfe von SAML und SOAP. Auf diese Sub-Protokolle und Möglichkeiten wird im folgenden kurz eingegangen.

Web Redirects Web Redirects werden bei Liberty über den HTTP Befehl 302 (Temporary Redirect) oder über HTML-Form POST basierten Redirects realisiert. In Abbildung 3 ist dargestellt wie Web Redirect funktioniert. Der Browser des Benutzers schickt eine HTTP Anfrage z.B. an den Service Provider (1). Als Antwort (2) bekommt er den Status Code 302 mit einer neuen URI gesendet. In dieser URI sind weitere Parameter enthalten, unter anderem auch die Rück-

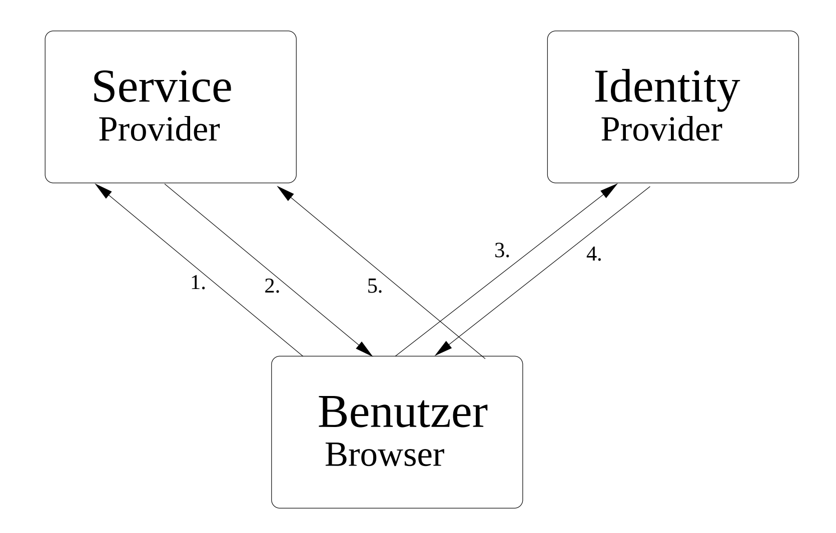


Abbildung 3: Web Redirects

kehradresse des SPs. Vom Browser wird erwartet, dass er der Aufforderung nachkommt und die angegebene URI aufruft (3). Der Identity Provider kann aus den Informationen in der URI erkennen, dass die Anfrage ursprünglich von dem Service Provider kommt und kann dem Browser wiederum über den Status Code 302 (4) zurück zum Service Provider senden (5) und zusätzlich weitere Informationen übergeben. Somit ist mit Hilfe des Browsers ein Kommunikationskanal zwischen SP und IDP aufgebaut.

Beim POST basiertem Redirect bekommt der Browser ein HTML Formular übertragen (2), bei dem der Submit-Button mit einer URI mit Paramtern verknüpft ist. Wenn auf SUBMIT geklickt wird, bekommt der empfangende Server die URI mittels der HTTP POST Methode gesendet (3) und kann den ursprünglichen Absender erkennen und dem Browser wiederum ein HTML Formular schicken (4) und seine Parameter in der URI mit einschließen. Der Browser macht wiederum ein HTTP POST (5) und kontaktiert den ersten Server. Über die Einbindung eines JavaScripts in das Formular kann der Vorgang automatisiert werden, sodass der Benutzer nicht auf SUBMIT klicken muss.

Nach [LAP03c] ist Web Redirection kein ideales Kommunikationssystem und hat ein paar gut dokumentierte Sicherheitsprobleme. Die Nachrichten werden im Klartext übertragen und sind somit abhörbar. Als Absicherung ist hier eine Kanalabsicherung wie SSL nötig. Diese Absicherung löst aber nicht das Problem, dass die Informationen auf der Browser.Seite entschlüsselt werden und im Klartext vorliegen. Diesem kann man nur entgegenwirken, indem man die Informationen schon vor dem Senden verschlüsselt. Eine weitere Beschränkung besteht durch die Gesamtlänge der URI. Es können nicht beliebig viele Informationen über diesen Kanal ausgetauscht werden. Gerade bei mobilen Geräten ist die Länge der URI recht klein. Diese und weitere Einschränkungen sind bei dem Design des Liberty Protokoll beachtet und in [RW03a] betrachtet.

Cookies Da das HTTP-Protokoll ein zustandloses Protokoll ist, muss jede Anfrage an einen Server als Einzelanfrage bearbeitet werden, obwohl vielleicht mehrere direkt hintereinander ausgeführt worden sind. Um eine Möglichkeit einer Zustandspeicherung in das HTTP-Protokoll integrieren zu können, sind die Cookies eingeführt worden. Die Cookies werden vom Browser automatisch an den Server bei einem Verbindungsaufbau gesendet und können vom Server gelesen und geändert werden, sodass der Server in diesen Cookies Informationen ablegen kann. Nach [Eck03] sind Cookies zur Erleichterung des elektronischen Einkaufens entwickelt worden. Ein Cookie soll als Warenkorb dienen, in dem die ausgewählten Artikel gespeichert werden können. Cookies dienen einem Server somit zur Speicherung von Zustandsinformationen auf der Browserseite.

Allerdings sind viele Browser aus sicherheitstechnischen Gründen so konfiguriert, dass sie Cookies entweder überhaupt nicht zulassen oder nur auf den Server beschränken der das Cookie gesetzt hat. Diese Beschränkung basiert dabei auf dem DNS-Namen des Servers. Somit ist es nicht möglich, dass ein zweiter Server mit einem anderen DNS-Namen auf das Cookies zugreifen kann. Wenn es in einem Circle of Trust mehrere IDPs gibt, könnte man in einem Cookie speichern, welcher IDP vom Benutzer bevorzugt wird. Dieses Problem wird als das Introduction Problem bezeichnet (siehe [LAP03c]). Da aber ein Cookie nicht von einem zweiten Server mit unterschiedlichem DNS-Namen gelesen werden kann, kann ein zweiter SP nicht von einem anderen SP erfahren, welcher IDP der Standard IDP des Benutzers ist. Eine Möglichkeit, dieses Problem zu lösen ist, dass alle Dienste in dem Circle of Trust eine Unterdomains zu einer gemeinsamen Hauptdomain besitzen. Also z.B. car.airline.com und room.airline.com. Somit könnte ein gemeinsames Cookie unter airline.com angemeldet werden, auf das alle zugreifen können. Da aber mittlerweile die Internetadresse ein fester Bestandteil einer Firma ist, kann man nicht davon ausgehen, dass die Dienste in einem Circle of Trust unter einer Hauptdomain zusammenfassbar sind.

Aufgrund der gerade geschilderten Probleme mit Cookies ist der Cookie-basierte Ansatz nur optional im Liberty Protokoll vorhanden. Um das Introduction Problem zu lösen wird entweder ein Standard IDP fest vorgegeben oder aber dem Benutzer ein Liste der IDPs präsentiert, aus der er sich dann seinen favorisierten IDP auswählen kann.

Authentication Assertion Damit sich ein Principal bei einem Service Provider ausweisen kann, muss er eine Authentication Assertion von einem Identity Provider vorweisen. Diese wird nach ID-FF in einer SAML 1.1 Nachricht verpackt. Das SAML (Security Assertion Markup Language) Protokoll definiert nach [HM04] ein Framework, um Sicherheitsinformationen zwischen Online-Geschäftspartnern auszutauschen. SAML definiert dabei drei Typen von Assertions: Authentication Assertion sagt aus, dass ein Subjekt authentifiziert wurde, Attribute Assertion spezifiert gewisse Details eines Benutzers und Authorization Decision

gibt an was ein Benutzer machen darf. Beim ID-FF Protokoll wird nur die Authentication Assertion verwendet und dient dazu, dass ein SP eine Anfrage an den IDP stellen kann und vom IDP eine Antwort bekommt, in der beschrieben ist, welcher Principal zu welchem Zeitpunkt mit welcher Methode authentifiziert wurde. Diese Authentication Assertion ist dabei die Antwort (Nr. 6/7 in Abbildung 2) auf ein Authentication Request vom SP.

Abbildung 4: Authentication Assertion

Wie man in Abbildung 4 sieht, besteht eine Assertion dabei aus einer Assertion ID, dem Namen des Ausstellers dieser Assertion (IDP), dem Zeitpunkt wann diese Assertion ausgestellt wurde, dem Zeitpunkt bis wann diese Assertion gültig ist, für wen sie bestimmt ist (Audience Restriction), der Methode wie sich der Principal authentifiziert hat (z.B. Passwort), dem Zeitpunkt der Authentifizierung, dem Pseudonym auf der SP und IDP Seite und einer digitalen Signatur der Assertion. Diese SAML verpackten Assertions werden entweder über Web Redirects oder über das SOAP-Protokoll ausgetauscht.

Das SOAP (Simple Object Access Protocol) ist nach [W3C00] ein Protokoll um in einer dezentralisierten, verteilten Umgebung Informationen auszutauschen. Es baut auf dem XML Standard auf und kann prinzipiell über verschiedenste Protokolle übertragen werden. Es wird bei Liberty ID-FF aber nur über das HTML Protokoll übertragen.

In Abbildung 5 ist zu erkennen, dass eine SOAP Nachricht in eine HTTP Nachricht eingebettet ist. Eine SOAP Nachricht besteht aus einem SOAP Header und einem SOAP Body. Innerhalb dieses Bodys können die Nachrichten übertragen werden. Im Falle des Liberty Protokolls ist die Authentication Assertion spezifiziert nach SAML.

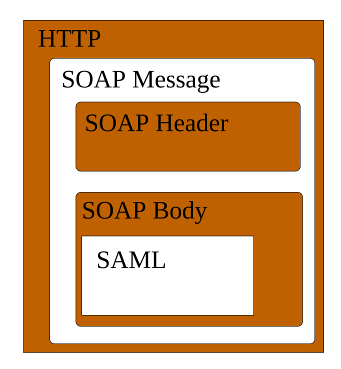


Abbildung 5: Struktur einer SOAP/SAML Nachricht

AuthnContext Ein essentieller Bestandteil des Liberty Protokolls ist die genaue Angabe der Authentifikationsmethode. Diese Angabe muss sowohl vom Service Provider als auch vom Identity Provider getätigt werden. Der Service Provider gibt dem Identity Provider vor, welche Authentifikationsmethoden er akzeptiert. Der Identity Provider wählt sich aus den vorgegebenen Methoden eine Methode heraus und führt diese mit dem Subjekt durch. Der IDP muss dem SP in seiner Antwort genaue Angaben machen mit welcher Methode er das Subjekt authentifiziert hat. Die Spezifikation dieser Authentifikationsmethoden ist in der Liberty ID-FF Authentication Context Specification [LAC03] des Liberty Protokolls angegeben. Diese Spezifikation soll es in erster Linie den Service Providern ermöglichen, die vom Identity Provider durchgeführte Authentifikationmethoden bewerten zu können. Der Identity Provider kann mit Hilfe dieser Spezifikation genaue Angaben über den initialen Identifizierungmechanismus, den eingesetzen Mechanismus zur Verhinderung von Kompromittierung von Berechtigungsnachweisen bzw. die Art der Speicherung der Nachweise machen. Außerdem können genaue Angaben über die Authentifikationsmechanismen und deren Absicherung gemacht werden. Anhand dieser Angaben ist es dem Service Provider möglich, eine Sicherheitsbewertung durchzuführen, inwieweit er der vom Identity Provider angegebenen Authentifikation vertrauen kann.

Damit nicht alle Details des AuthnContext angegeben werden müssen, um eine Kommunikation zwischen Identity Provider und Service Provier zustande kommen zu lassen, wurde die Spezifikation um sogenannte **Authentication Context Classes** erweitert. Diese unterteilen die verschiedenen Mechanismen der Authentifikation in gewisse Klassen und werden über festgelegte URIs spezifiziert. Standardmäßig gibt es zum Beispiel die Klasse „Password" in der alle Verfahren zusammengefasst sind, die eine Passwortauthentifikation über einen unverschlüsselten Kanal durchführen oder z.B. die Klasse „PasswordProtectedTransport" in der die Verfahren einer Passwortauthentifikation über eine geschützte Verbindung zusammengefasst sind. Mit diesen Klassen wird die Sicherheitsbewertung des Service Providers vereinfacht, da die gleichen Mechanismen in einer Klasse ähnliche Sicherheitskriterien mit sich bringen. Außerdem wird es dem Service Provider

vereinfacht, seine bevorzugten Authentifikationsmethoden anzugeben. Der Identity Provider kann über diese Spezifikation seine Authentifikationsfähigkeiten auf einfachem Weg veröffentlichen.

Die Erstellung und Überprüfung des AuthnContext wird von SourceID (s.u.) nicht übernommen, sondern muss von der Applikation auf beiden Seiten, also sowohl auf der Identity Provider Seite als auch der Service Provider Seite, implementiert werden. Da ein Bestandteil dieser Diplomarbeit die Authentifikation über ein Hardware-Token(Kapitel 5.3) ist, muss die Angabe der benötigten Authentifikation auf Seiten des Service Provider von der Applikation erstellt werden und nach Erhalt der Bestätigung überprüft werden. Auf Seiten des Identity Providers muss eine entsprechende Authentifikationsanfrage erst dahingehend überprüft werden, ob eine gefordete Methode durchgeführt werden kann und das detaillierte Ergebnis als Authentication Context zurückgeschickt werden.

2.5 Sicherheitsbetrachtung

Da die Nachrichten des Liberty Protokolls über das Internet übertragen werden, müssen die Sicherheitsprobleme des Protokolls und der benutzten Sub-Protokolle genau betrachtet werden. Nach [Eck03] gehören zu einem richtigen Protokolldesign die Beachtung folgender Sicherheitsaspekte:

- Vertraulichkeit
  die Unterbindung unautorisierter Informationsgewinnung
- Authentizität
  die Echtheit und Glaubwürdigkeit eines Subjekts
- Integrität
  das Verhindern von unautorisiertem und unbemerkten manipulieren geschützter Daten
- Frische
  Verhindert die Wiedereinspielung von alten abgefangenen Nachrichten

Nach diesen Aspekten wird nun das Liberty Protokoll bewertet.

In der Spezifikation zum SAML Protokoll [HM04] wird festgelegt dass, wenn die Integrität und die Vertraulichkeit einer Nachricht benötigt wird, die Verwendung von HTTP über SSL 3.0 oder TLS 1.0 gefordert (RECOMMENDED) wird. Außerdem wird empfohlen, wenn eine abhängige Partei (SP) von Authentifikationsdaten einer anderen Partei (IDP) abhängig ist, die Benutzung von SSL 3.0 mit Server und Client Zertifikaten anzuwenden. Wenn eine Assertion über ein web redirect über eine dritte Partei, z.B. den Browser des Benutzers, geleitet wird, muss diese mit einer digitalen XML-Signatur signiert werden. Da es mittlerweile einige Angriffsszenarien auf Single-Sign-On Protokolle auf Basis von SAML gibt, empfiehlt [Gro03] immer einen sicheren Übertragungskanal wie SSL

3.0 oder TLS 1.0 mit mindestens einseitiger Authentifikation für die Übertragung der Informationen zu verwenden. SSL bietet Authentizität, Frische und verhindert Wiedereinspielungen von Nachrichten, sodass ein Großteil der Angriffsmöglichkeiten abgefangen werden.

Nach [TC03] unterscheidet Liberty die Absicherung des Protokolls auf der Kommunikationsebene (Channel Security) und auf der Nachrichtenebene (Message Security).

- **Channel Security**
  Mit der Channel Security wird dabei die Absicherung der Kommunikation zwischen Kommunikationspartnern addressiert, im Falle von Liberty die Komunikation zwischen Identity Provider, Service Provider und dem Browser des Benutzers. Liberty gibt hier vor, dass die Absicherung der Kommunikation mittels TLS1.0 oder SSL3.0 geschehen muss. Die Spezifikation von Liberty besagt, dass für die Channel Security Vertraulichkeit und Datenintegrität vorgeschrieben sind, welche SSL bzw. TSL bieten. Außerdem muss bei Verwendung von SSL zum Zwecke der Authentifikation mindestens der Identity Provider ein serverseitiges Zertifikat vorweisen, dass über die PKI-Infrastruktur überprüft werden kann. Der IDP kann darüber hinaus auch ein client-seitiges Zertifikat verlangen.

- **Message Security**
  Die Message Security behandelt die Absicherung der Liberty Protokoll Nachrichten zwischen Identity Provider, Service Provider und dem Browser des Benutzers, die über den gerade genannten Kommunikationskanal übertragen werden. Teile der Liberty Protokoll Nachrichten müssen signiert werden und vom Empfänger validierbar sein. Liberty spezifiert nicht, mit welcher Methode das zu geschehen hat, nur dass es geschehen muss. Nach [TC03] gewährleistet das Signieren und Validieren der Signatur die Datenintegrität, die Herkunft der Daten und bietet eine Basis für die Nichtabstreitbarkeit einer Nachricht.

Jeder Teilnehmer in einem Circle of Trust muss bei den Partnern fest vorab eingetragen worden sein, damit über das Liberty Protokoll kommuniziert werden kann. Hierbei wird, wie z.B. bei SourceID (s. u.), nicht nur der Name und die Adresse des Partners eingetragen, sondern auch der öffentliche Schlüssel des X509-Zertifikats des Partners. Dieses Zertifikat wird von jedem Teilnehmer nur für den Zweck der Liberty Nachrichten erstellt und kann selbst signiert sein. Der öffentliche Schlüssel des Zertifikats wird an die Partner im Circle of Trust weitergegeben, sodass diese von anderen Teilnehmern signierte Nachrichten validieren können.

Um einen höheren Schütz gewährleisten zu können, müssen die IDPs und SPs Schlüsselpaare für die Message Security verwenden, die sich von denen zur Kom-

munikationskanalabsicherung (Channel Security) unterscheiden. Außerdem wird erwartet, dass Nachrichten gegen Replay-Attacken geschützt werden und empfangene Antworten daraufhin überprüft werden, dass sie passend zu den gestellten Fragen sind. Hierzu sollten zeitbasierte Nachrichten verwendet werden um die Frische nachweisen zu können.

Zusammengefasst muss für Nachrichten des Liberty Protokoll nach der Message Security Richtlinie folgendes gelten:

- Per-message data integrity
- Transaction integrity
- Data origin authentication (Quellursprung)
- Non-repudiation (Nicht Abstreitbarkeit)
- Confidentiality (optional)

Da die verwendeten Teilprotokolle wiederum weitere Angriffspunkte haben, ist eine abschließende Sicherheitsanalyse des Liberty Protokolles schwer möglich. Auf das SSL Protokoll beispielsweise gibt es Angriffe, die auf der DNS-Struktur oder der Veränderung der Systemuhrzeit aufbauen können.

2.6 SourceID

SourceID wurde im Jahre 2001 gegründet und wird von Ping Identity, HP, Nokia und General Catalyst gesponsort und betrieben. Das Ziel von SourceID ist es Open-Source Tools, Applikationen und Infrastruktur aufzubauen, um Federated Identity Managment unter Verwendung von SourceID zu ermöglichen. SourceID benutzt dabei offene Protokolle und Standards, dazu gehören die Standards des Liberty Alliance Projekts, der OASIS SAML Gruppe und die WS-* Initiative von IBM und Microsoft. Es soll dem Entwickler von Software erleichtert werden (siehe [FE03]), seine Software um Identity Federations zu erweitern, da er sich nicht mehr um die Details des Liberty Protokoll und der darunterliegenden Protokolle, wie XML, XML Digital Signatures, SOAP und SAML kümmern muss. SourceID hat bisher mehrere Versionen ihres Frameworks veröffentlicht. Dazu gehört **SourceID.Java**, **SourceID.NET** sowie **SourceID Liberty**. SourceID.Java und SourceID.NET implementieren dabei das Liberty Protokoll ID-FF der Version 1.1. SourceID Liberty das Liberty ID-FF Protokoll der Version 1.2. Alle drei Frameworks wurden von der Liberty Alliance getestet und als konform bezeichnet und mit dem Prädikat „Liberty Alliance Interoperable" ausgezeichnet. Alle drei Frameworks sind Open-Source Projekte, die nach einer Open-Source oder Public-Source Lizenz freigegeben sind (vgl. [sou03]).

SourceID.NET ist eine Erweiterung zur Integration des Liberty Protokolls für das .NET Framework von Microsoft. SourceID.Java ist die Erweiterung für

die Java Programmierumgebung und zielt auf die Integration in eine Java-Web-Applikation, die innerhalb eines Containers wie Tomcat oder JBoss läuft. SourceID Liberty ist das neueste Framework von SourceID und zielt zusätzlich zur Integration in eine Java-Web-Applikation auch noch die Integration als autonomer Server an, damit dieser innerhalb vieler unterschiedlicher Programmsprachen/Umgebungen einbindbar ist. Für die Zukunft ist bei SourceID geplant, dass ID-FF Version 1.2, ID-WSF sowie die WS-Federation Spezifikation zu unterstützen.

SourceID.Java bietet eine API zur Kommunikation mit anderen Teilnehmern eines Circle Of Trust und übernimmt die Handhabung der SOAP, SAML und XML Nachrichten Auerdem bietet eine high-level Schnittstelle für die Applikation (siehe Abbildung 6) und ist als ein zusätzliches Modul in einer J2EE Umge-

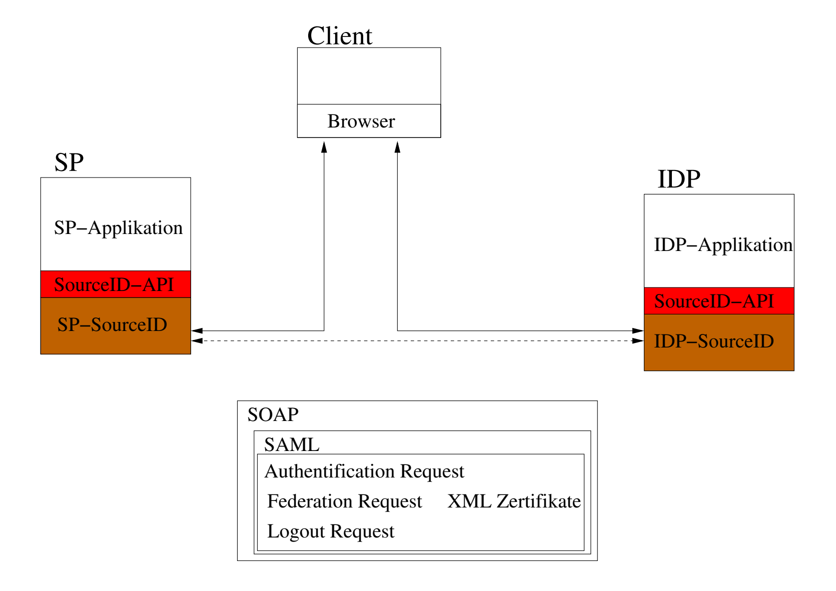


Abbildung 6: SourceID-API

bung gedacht. Eine typische Anwendung für die Erweitung mit SourceID ist eine Webapplikation, die innerhalb eines Webcontainers, wie z.B. Tomcat oder JBoss, läuft, wobei der komplette Versand und die Überprüfung der Liberty Nachrichten von SourceID übernommen wird. Der Programmierer einer Anwendung muss sich nicht mit den Details des Liberty Protokolls und deren Unterprotokolle beschäftigen. Die Logik des Liberty Protokolls muss allerdings trotzdem in die Applikation integriert werden. Die Applikation muss als Service Provider oder Identity Provider in den Konfigurationsdateien von SourceID eingerichtet werden und an die SourceID-API Schnittstellen angepasst werden. Die Requests, die im Liberty Protokoll möglich sind, müssen von der Applikation sowohl selbst initiiert als auch entsprechende Antworten von ihr verarbeitet werden können.

Damit eine Anwendung mit Hilfe von SourceID an einem Circle of Trust teilnehmen kann muss einiges vor dem Aufruf der API konfiguriert werden.

- In der Datei **web.xml** werden die Schnittstellen von SourceID-Java eingetragen. Diese Schnittstellen werden sowohl von den anderen Teilnehmern des Circle of Trust angesprochen als auch direkt von der Applikation und müssen somit direkt aus einem Browser heraus erreichbar sein. Beispielhaft ist in Listing 1 der Eintrag des SourceID-SSO-AuthnRequestor abgebildet. Die nach außen hin offene Adresse ist die Adresse der Applikation an sich plus den Zusatz „/sso/authnRequest”,
z.B.: http://localhost:8080/jpetstore/sso/authnRequest.

```
<servlet>
        <servlet-name>SourceID-SSO-AuthnRequestor</servlet-name>
        <description>Provides Service Provider (SP) Authentication Request
            Services</description>
        <servlet-class>org.sourceid.sso.servlets.AuthnRequestor</servlet-class
            >
</servlet>

<servlet-mapping>
        <servlet-name>SourceID-SSO-AuthnRequestor</servlet-name>
        <url-pattern>/sso/authnRequest</url-pattern>
</servlet-mapping>
```

Listing 1: Auszug aus der web.xml

  Wenn eine Service Provider Applikation eine Authentifikation mit SourceID initiieren will, verweist sie den Browser des Benutzer weiter auf die Adresse des SourceID-SSO-AuthnRequestor. Ein Beispiel hierzu befindet sich in Listing 22 auf Seite 99.

- In der Datei **sourceid-sso.xml** wird spezifiziert, ob sich die Applikation in der Rolle eines SP oder IDP befindet und das eigene Zertifikat, das für das Signieren der Liberty Nachrichten nötig ist, sowie der UserID-Variablenname gespeichert. Es wird davon ausgegangen, dass die Applikation die Überprüfung, ob ein Subjekt sich authentifiziert hat, über das Vorhandensein einer UserID-Variable in dem Session Context der Applikation vornimmt. Somit hat SourceID-Java die Möglichkeit, bei einem Überschreiten des Zeitlimits oder bei einem Ausloggen, das von außen getriggert ist, diese UserID-Variable zu löschen und der Applikation mitzuteilen, dass das Subjekt nicht mehr eingeloggt ist.

- In der **sourceid-sso-providers.xml** werden alle beteiligten Partner innerhalb des Circle of Trust eingetragen.

```
<lib:SPDescriptor xmlns:lib="http://projectliberty.org/schemas/core/2002/12" xmlns:ds=
      "http://www.w3.org/2000/09/xmldsig#">
    <lib:ProviderID>jpetstore</lib:ProviderID>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Certificate xmlns:ds="http://www.w3.org/2000/09/xmldsig#">MIICaDCCAd
            ......</ds:X509Certificate>
        </ds:X509Data>
    </ds:KeyInfo>
    <lib:AssertionConsumerServiceURL>http://localhost:8080/jpetstore/sso/authnRequest
        </lib:AssertionConsumerServiceURL>
```

```
    <lib:SoapEndpoint>http://localhost:8080/jpetstore/sso/soap/endpoint</lib:
        SoapEndpoint>
    <lib:SingleLogoutServiceURL>http://localhost:8080/jpetstore/sso/logout</lib:
        SingleLogoutServiceURL>
    <lib:SingleLogoutServiceReturnURL>http://localhost:8080/jpetstore/sso/logout</lib:
        SingleLogoutServiceReturnURL>
    <lib:FederationTerminationServiceURL>http://localhost:8080/jpetstore/sso/fedterm</
        lib:FederationTerminationServiceURL>
    <lib:FederationTerminationServiceReturnURL>http://localhost:8080/jpetstore/sso/
        fedterm</lib:FederationTerminationServiceReturnURL>
    ...
    ...
    ...
</lib:SPDescriptor>
```

Listing 2: Auszug aus der sourceid-sso-providers.xml

Wie in Listing 2 zu sehen, werden die eindeutigen Namen der Partner mit dem Punkt <lib:ProviderID> sowie deren öffentlichen Schlüssel ihrer Zertifikate unter <ds:KeyInfo> gespeichert. Außerdem werden die Adressen der Schnittstellen der beteiligten Partner gespeichert. Somit ist sichergestellt, dass keine unautorisierte Person sich als ein Beteiligter des Circle of Trust ausgeben kann, da zu jedem beteiligtem Partner dessen öffentliches Zertifikat gespeichert ist. Dies bringt den Nachteil, dass Änderungen immer von Hand eingetragen werden müssen.

```
<account-handler>org.diplom.idp.AccountHandlerJDBC</account-handler>
```

Listing 3: Auszug aus der sourceid-sso.xml

Zusätzlich zu den Informationen in den angegebenen xml-Dateien verlangt SourceID.Java nach einem AccountHandler. Dieser AccountHandler (siehe Listing 3) ist eine Java-Klasse, die von SourceID.Java aufgerufen wird und eine Schnittstelle zwischen der lokalen Benutzerdatenbank und SourceID darstellt. SourceID überprüft bei von Partnern im Circle of Trust ankommenden Nachrichten, ob die Subjekte, die von den Partnern authentifiziert wurden in der lokalen Datenbank gespeichert sind und es Federationeinträge mit den Subjekten und der ID der Partners gibt. Über den AccountHandler wird diese Zuordnung der Pseudonyme zu den lokalen Benutzerkennungen vorgenommen.

In Bild 7 wird verdeutlicht, wie der Ablauf der Kommunikation zwischen der SourceID-API und der Applikation auf Seiten eines Service Providers von statten geht. Die Applikation initiiert (1) eine Authentifikationaufforderung eines Subjektes und erstellt dafür ein Authentification Request, der die Art der benötigten Authentifikation sowie den Identity Provider angibt, der die Authentifikation durchführen soll, sowie die Rücksprungadressen zu der Applikation bei gelungener Authentifikation (Success-Page) und im Fehlerfall (Failure-Page) angibt. Dieser Authentification Request wird dann an den SourceID-SSO-Authenticators übermittelt (2). SourceID übernimmt jetzt die Anfrage und nimmt die Kommunikation mit dem angegebenen Identity Provider über web redirects auf. Falls die Authentifikation positiv verlief, überprüft SourceID-Java über den Aufruf (3) des AccountHandlers, ob es zu dem vom IDP angegeben Pseudonym einen Eintrag

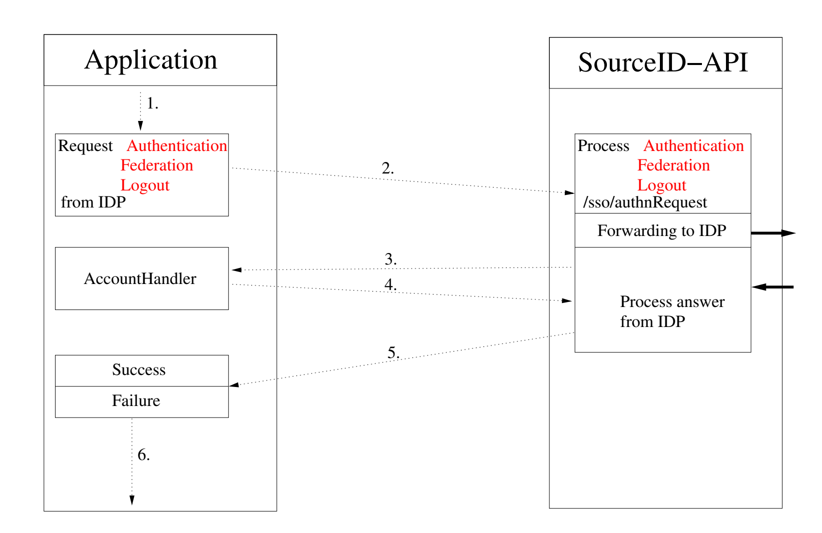


Abbildung 7: SP: Interaktion Applikation ⇔ SourceID-API

in der Datenbank gibt. Der AccountHandler liefert im positiven Fall die lokale Benutzerkennung des Subjekts zurück (4). Danach wird die Kontrolle wieder zurück an die Applikation gegeben (5). Hierbei wird je nach gelungener oder nicht gelungener Authentifikation die Success-Page oder Failure-Page der Applikation aufgerufen. Im Fehlerfall bekommt die Applikation eine Fehlermeldung übermittelt, warum die Authentifikation fehlgeschlagen ist.

Das Anlegen einer neuen Federation zwischen einem Account bei der Applikation und einem Account bei einem Identity Provider funktioniert dabei auf die gleiche Weise wie in Abbildung 7 über den SP, nur wird hierbei am Anfang ein Federation Request zusammengestellt. Dieser beinhaltet den Namen des beteiligten Identity Providers, die Rücksprungadressen, sowie die Kennung des lokalen Benutzers. Dieser Request wird dann an SourceID-Java übergeben (2). Daraufhin generiert SourceID-Java ein Pseudonym für den Benutzer und nimmt die Kommunikation mit dem Identity Provider auf. Die Applikation bekommt zum Schluss wieder über die Success und Failure Page mitgeteilt, ob die Föderation zustande kam. Der Aufruf zur Beendigung eben dieser - der Terminate Federation Request - verläuft dabei analog zum Federation Request.

Der Aufruf zum Logout beim Identity Provider wird über ein Logout Request an SourceID-Java übergeben. Hierbei wird der Name des Identity Providers, die lokale Benutzerkennung sowie die Rücksprungadressen an SourceID-Java übermittelt. Die Applikation bekommt daraufhin über die Success oder Failure Page mitgeteilt, ob der Global Logout funktioniert hat.

Auch für die IDP Seite des Liberty Protokolls gibt es von SourceID.Java eine API. Diese ist geringfügig anders als die API der SP Seite. Die Konfiguration wird genauso durch die oben genannten drei Dateien spezifiziert. Die weiteren Mitglieder des Circle of Trust werden in der Datei **sourceid-sso-providers.xml**,

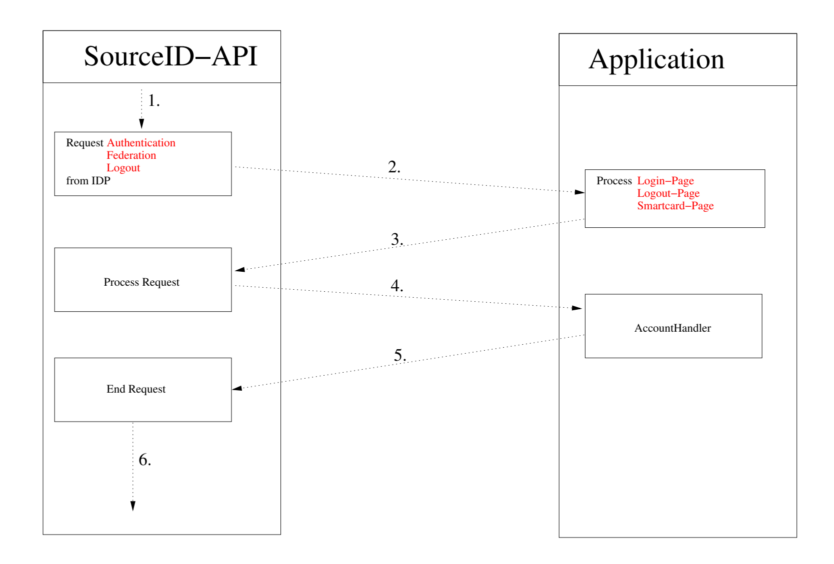


Abbildung 8: IDP: Interaktion Applikation ⇔ SourceID-API

die nach außen offenen Schnittstellen in der **web.xml**, die Angabe über den AccountHandler und die IDP-Rolle von sourceID wird in der **sourceid-sso.xml** angegeben.

Der Ablauf der Kommunikation zwischen der SourceID.Java-API und der Applikation auf der IDP Seite ist in Abbildung 8 dargestellt. Im Normalfall kommt eine Anforderung von einem SP an den IDP zur Authentifikation eines Subjekts. Die Aufforderung(1) geht dabei direkt an die in der web.xml spezifierten offenen Schnittstelle von SourceID-Java. Dort wird sie überprüft und die Kontrolle an die Login-Seiten der IDP-Applikation übergeben(2). Hier führt die Applikation je nach Anforderung im Authentication Request des SP die Authentifikation durch. Hiernach wird die Kontrolle wieder an SourceID mit der Angabe der lokalen Benutzerkennung übergeben(3). SourceID überprüft nun, ob es eine federation zwischen der lokalen Benutzerkennung und der Benutzerkennung auf der anfragenden SP-Seite gibt. Hierfür wird der AccountHandler der Applikation aufgerufen(4). Der AccountHandler überprüft die lokalen Datenbank nach der eingetragenen federation und übergibt das Ergebnis wieder zurück an SourceID(5). SourceID generiert nun die Antwort für den SP und trägt als Benutzerkennung das Pseudonym ein, das ihm der AccountHandler übergeben hat. Falls ein Subjekt direkt auf die Seiten des IDP zugreift, gelangt er direkt auf die Login-Seiten der Applikation. Hierfür wird keinerlei Aufruf der SourceID-API durchgeführt, da sich der Benutzer nur lokal auf den Seiten des Identity Providers befindet.

3 Aspektorientierte Programmierung

Ein zweiter Baustein, der in dieser Diplomarbeit verwendet wird, ist die aspektorientierte Programmierung (AOP). Wie im folgenden Kapitel beschrieben wird, eröffnet diese Programmiertechnik völlig neue Wege in der Modularisierung von Software. Die später beschriebene Möglichkeit, ein Programm nach dessen Compilation um Funktionalität zu erweitern, ermöglicht eine starke Unabhängigkeit in der Entwicklung eines Moduls, welches die sicherheitskritischen Teile beinhaltet.

Ein Softwaresystem besteht aus verschiedenen Aufgaben, die durch die Software ausgeführt werden sollen. Hierbei kann grob zwischen „Core Concerns" und „Crosscutting Concerns" unterschieden werden. Ein „Core Concern" enthält die eigentliche Funktionalität eines System, wohingegen ein „Crosscutting Concern" sich meist um betriebssystemnahe Aufgaben oder IO kümmert. Als ein Beispiel kann die Loggingproblematik gesehen werden.

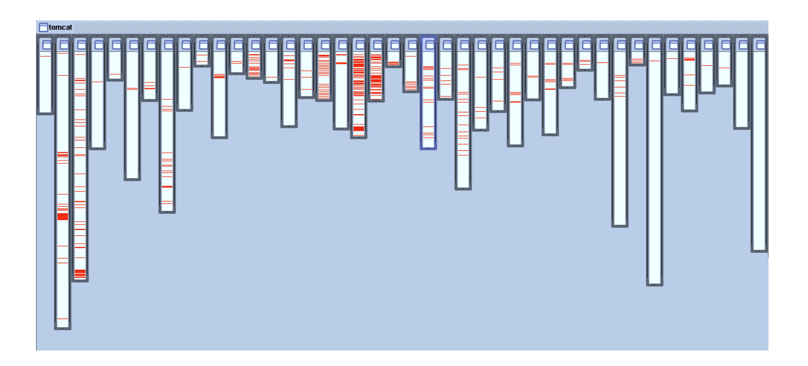


Abbildung 9: Einsatz von Logging im Tomcat

Die Aufgabe die sich stellt ist: „Verzeichne jeden Zugriff auf eine Methode und schreibe dies in eine Datei". Diese an sich einfache Aufgabenstellung bedeutet für den Programmierer in der Entwicklung, dass er in jede Methode Sourcecode einbinden muss. Am Beispiel des Tomcat Application Server wird in Abbildung 9 gezeigt, wie stark diese Verteilung ist. Die senkrechten Balken repräsentieren die Klassen wohingegen die waagerechten für die Aufgabe Logging stehen. Daneben bedeutet dies für den Programmierer, dass er an jedem dieser Punkte immer wieder die gleichen Befehle im Code einfügen muss. Die wichtigsten hierbei sind:

- Der Importbefehl: `import org.apache.commons.logging.*;`,
- die Initialisierung der Logklasse:
  `private static Log log = LogFactory.getLog( example.class );` und
- in jeder Methode das Ausführen des loggens: `log.debug(''Entering: [ Methode xyz ]'');`

Falls nun eine Änderung in der Loggingstrategie gewählt wird, z.B. logp statt log, so muss der gesammte Sourcecode des Projektes durchsucht werden, um diese Änderung durchzuführen. Dies ist sehr fehleranfällig, da es in größeren Projekten durchaus langwierig ist, alle Stellen zu finden, die wirklich verändert werden müssen und grep nur manchmal hilft.

Aufgrund der Tatsache, dass es offensichtlich Module gibt, die durch den objektorientierten Ansatz nicht isoliert werden können, wurde eine Erweiterung der Objektorientierung entwickelt. Diese versucht, Aspekte in einer Software wie z.B. Logging zu isolieren und in einer kompakten Form zu beschreiben.

In dieser Arbeit wird der Begriff des Aspekts mit zwei verschiedenen Interpretationen verwendet. Zum einen kommt der Aspekt in seiner normalsprachlichen Bedeutung zum Einsatz. Zum anderen wird er auch im Zusammenhang mit dem programmiersprachlichen Konzept aus AOP verwendet. Hierbei wird zur besseren Unterscheidbarkeit die englischen Schreibweise (Aspect) verwendet. Dieser Aspect wird in Kapitel 3.1.1 vorgestellt.

Diese aspektorierierte Programmierung hat ihren Ursprung im Xerox/Alto Reserch Center (Das älteste Dokument stammt aus dem Jahr 2000 [Voe00]). Die Forschungsarbeiten begannen mit den Dokumenten [KLM+97] und [MKL97], die aus dem Jahr 1997 stammen.

Es wird in dieser Diplomarbeit AspectJ als Implementierung der aspektorientierten Programmierung verwendet. Die Wahl wurde aufgrund der guten Dokumentation und der hohen Produktreife getroffen.

3.1 Die Sprache

Jede Aufgabe, die das System später duchführen soll, wird im Rahmen von AOP als ein Concern wahrgenommen. Bei einem Webshop sind zum Beispiel das Hinzufügen von Artikeln zum Warenkorb und das Entfernen von Artikeln, Gesamtsumme berechnen und Protokollieren von Handlungen Concerns. Hierbei wird nun zwischen Systemconcerns und Cross-Cutting Concerns unterschieden. Der Systemconcern stellt eine spezielle, sich nicht wiederholende Aufgabe dar, die von dem System erledigt werden soll. Diese werden durch die objektorientierte Programmierung in Module aufgeteilt. Der Cross-Cutting Concern hingegen lässt sich durch objektorientierte Verfahren nur sehr schwer beschreiben, wie es am Anfangs gezeigten logging-Beispiel zu sehen ist. Ein Cross-Cutting Concern steht sozusagen orthogonal zur objektorientierten Methode und beschreibt Aufgaben, die sich an vielen Stellen wiederholen. Dies wird in Abbildung 10 illustriert.

In der Entwicklung einer Software gilt es diese Cross-Cutting Concerns zu isolieren, getrennt zu implementieren und dann wieder mit dem Programm zu verweben. Dieses Verweben der Eigenschaften wird durch einen sogenannten Waver durchgeführt.

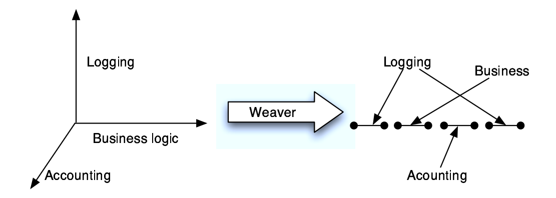


Abbildung 10: Zusammenfügen eines Systems aus Concerns

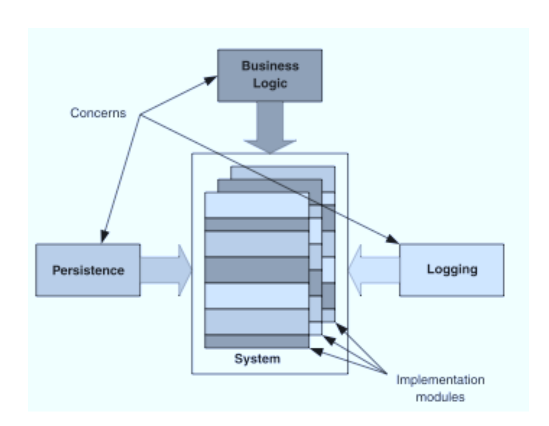


Abbildung 11: Ein System wird als Komposition von Concerns betrachtet [Lad03]

3.1.1 Die Syntax

Die Sprachdefinition von AspectJ unterscheidet Pointcuts, Advices, inter-type Declarations und Aspects. Ein Pointcut beschreibt, wo ein Aspect eingefügt werden soll. Der Advice sagt, wann der Aspect ausgeführt werden soll.

Join Point Der Join Point ist das fundamentalste Konzept der AspectJ-Definition. Durch ihn kann jeder identifizierbare Ausführungszeitpunkt innerhalb eines Programms beschrieben werden. Es werden folgende Join Points in AspectJ unterschieden:

Execution Der Execution Join Point bezieht sich auf die bezeichnete Methode an sich. Es wird die Ausführung der Methode gekapselt.

Call Hier sind alle Methoden selektiert, welche eine bestimmte Methode aufrufen. Im Unterschied zum vorherigen Join Point, wird jetzt nicht die Methode an sich sondern ihre Aufrufe gekapselt.

Constructor Beim Constructor Join Point handelt es sich um einen Call oder Execution Join Point, der auf den Constructor einer Klasse angewendet wird.

Field Access Ein Join Point, der die Lese- und Schreibzugriffe auf eine Speicherstruktur kapselt.

Exception Handler Execution Dieser liegt vor, wenn der Code eines Errorhandlers gekapselt wird.

Class Initialization Um Aktionen zum Ladezeitpunkt einer Klasse einzuweben, kann der Class Initialization Join Point verwendet werden.

Object Initialization Zum Zeitpunkt der Initialisierung, also der Erzeugung einer Klasse, greift dieser Join Point.

Object Pre-Initialization Kurz bevor eine Klasse erzeugt wird, wird dieser Join Point gesetzt. Normalerweise ist dies der Zeitpunkt, wenn super()-Aufrufe stattfinden.

Advice Execution Um einen Advice, als Ausführungszeitpunkt eines Advices, zu erfassen kann dieser Join Point verwendet werden.

R. Laddad zeigt in [Lad03, S. 50] weitere Beispiele und Anwendungsmöglichkeiten der verschiedenen Join Points.

Pointcut Durch einen Pointcut wird ein spezieller Join Point im Programmablauf definiert und mit einem Namen versehen. Ein Join Point besteht aus einem Pointcut-Type und einer Signatur. Der Pointcut-Type enthält einen Join Point, wie er weiter oben klassifiziert wird, und die Signatur beschreibt die Stelle, an der der Join Point angewendet werden soll. Ein einfacher Pointcut kann so aussehen:

```
pointcut anchor(HttpServletRequest request, HttpServletResponse response) : !within(
    HTMLRewriterAspect) && execution(void *._jspService(HttpServletRequest,
    HttpServletResponse)) && args(request, response);
```

Listing 4: Beispiel eines Pointcut

Der in Listing 4 gezeigte Pointcut ist dem HTMLRewriter-Aspekt (siehe 8.7) entnommen und selektiert jeden Aufruf der Methode, die in einer Webapplikation den HTML-Code erzeugt, der dem Client übertragen wird. Zusätzlich wird noch der **Context** des spezifizierten Join Points festgehalten. Dies bedeutet, dass innerhalb des Advice ein Zugriff auf die Variablen besteht, die auch die Methode übergeben bekommt.

Advice Der Advice ist der Code, der zu einem bestimmten Join Point ausgeführt werden soll. Daneben kann angegegben werden, zu welchem Zeitpunkt ein Aspect an den jeweiligen Join Point gebunden werden soll. Dazu stehen die folgenden drei Möglichkeiten zur Auswahl:

- Der Aspect wird vor (**before**) dem Joinpoint eingewebt.

- Nach (**after**) dem Jointpoint wird der Aspect eingewebt.
- Der Aspect kapselt den Aufruf (**around**).

Im Listing 5 wird ein before-Advice gezeigt, welcher jeweils vor dem Joint Point belowAnchor ausgeführt wird. Der after-Advice wird identisch dazu verwendet.

```
before() : belowAnchor(){
    log.debug("before()"+thisJoinPoint.toString());
}
```

Listing 5: Beispiel für before-Advice

Von diesem Schema unterscheidet sich der around-Advice. Dieser kapselt die durch den Join Point gewählte Methode. Erst durch den Proceed-Befehl wird die eigentliche Methode aufgerufen. Dies wird in Listing 6 gezeigt.

```
around() : belowAnchor(){
    log.debug("before()"+thisJoinPoint.toString());
    proceed();
    log.debug("after()"+thisJoinPoint.toString());
}
```

Listing 6: Beispiel für before-Advice

Introduction Durch eine Introduction können Veränderungen an Klassen, Interfaces und Aspecten eines Systems durchgeführt werden. Dies sind statische Veränderungen, die nicht direkt das Verhalten des Systems verändern. Es ist z.B. möglich eine Methode zu einer Klasse hinzuzufügen oder, wie in Listing 7 gezeigt, eine Klasse von einer anderen erben zu lassen.

```
    declare parents: Man implements Human;
```

Listing 7: Beispiel für eine Introduction

Compile-Time Declaration Eine andere statische Anweisung ist die Compile-Time Declaration. In Listing 8 wird für jede Verwendung der log-Methode eine Warnung beim Compilieren angezeigt.

```
declare warning: call(void *.log(..)) : "";
```

Listing 8: Beispiel für eine Compile-Time Declaration

Aspect Innerhalb von AspectJ nimmt der Aspect eine Position ein, welcher einer Klasse in Java vergleichbar ist. Sie beinhaltet Pointcuts, Introductions, Compile-Time Declarations und Advices. Daneben können noch Methoden, Variablen usw. wie in jeder normalen Javaklasse definiert werden.

Ein wichtiger Parameter bei der Erzeugung eines Aspects ist die „Aspect Association". Hierbei wird festgelegt, wann eine Instanz eines Aspects erzeugt wird. Hierbei unterscheidet AspectJ drei mögliche Kategorien:

- pro virtueller Maschine (die Standardeinstellung)
- pro Objekt
- pro Kontrolfluss

Die Standardeinstellung entspricht in Java einem Singleton. Es existiert nur eine Instanziierung des Aspects in der virtuellen Maschine. Pro Objekt meint, dass für jedes Objekt, mit dem ein Aspect verbunden wird, der Aspect instanziert wird. Die Pro-Kontrollflussassoziation (control-flow association) ermöglicht es, für einen Kontrollfluss Informationen übergreifend zu speichern. Diese Information steht in allen Point Cuts des Aspects innerhalb dieses Kontrollflusses zur Verfügung.

3.1.2 Der Weaver

Beim Verweben der Aspecte mit einem System können verschiedene Wege gegangen werden.

- Source Code Weaving
- Byte Code Weaving
- Weaving mithilfe eines class-Loaders

Beim Source Code Weaving wird der Sourcecode der Aspecte mit dem Sourcecode des Systems verwoben und dieser erzeugte Source wird dann durch den normalen Javacompiler übersetzt. Der zweite Ansatz erzeugt erst durch den Javacompiler Bytecode, der im nächsten Schritt durch den AspectJ-Weaver verwoben wird. Dies ist in Abbildung 12 dargestellt.

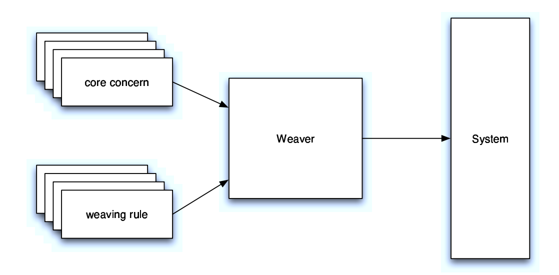


Abbildung 12: Schema des Weaver

Die dritte Möglichkeit unterscheidet sich stark von den bisher gezeigten Lösungen. Hier wird nicht das Programm verändert, sondern ein spezieller class-Loader eingesetzt, der das Weaving während der Programmausführung durchführt. Dieses Verfahren wird zum Beispiel vom Aspectwerkzframework als „hot deployment" ([Vas04]) bezeichnet. Wie auf der Webseite von Aspectwerkz [Vas04] hervorgehoben wird, besteht mit Hilfe dieses Konzepts die Möglichkeit, dynamisch Aspecte hinzuzufügen, zu entfernen oder zu ändern.

3.2 Einsatzbeispiele

Im folgenden wird durch einige Beispiele der Einsatz von AspectJ innerhalb der Diplomarbeit motiviert. Dabei soll gezeigt werden, wie stark ein Programm in seiner Struktur durch den Einsatz von AOP vereinfacht wird. Die Beispiele lehnen sich an die Beispiele von Ramnivas Laddad aus [Lad03] an.

3.2.1 Logging

Das Logging-Beispiel, das schon in der Einleitung zu diesem Kapitel Verwendung fand, soll nun durch einen Aspekt umschrieben werden. In der Diplomarbeit findet der Aspect wie in Listing 9 gezeigt Verwendung.

```
// import java.util.logging.*;
import org.aspectj.lang.*;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

aspect loggingAspect {

    private int test = 0;

    //private Logger _logger = Logger.getLogger("trace");
    private static Log log = LogFactory.getLog(loggingAspect.class);

    pointcut traceMethods() : execution(* tip.*.*(..)) && ! within(loggingAspect);

    loggingAspect() {
        test = 0;
    }

    before() : traceMethods()
    {
        test += 1;
        Signature sig = thisJoinPointStaticPart.getSignature();
        log.debug("Entering:␣[" + sig.getDeclaringType().getName()+"."+sig.getName()+"␣test:␣
            "+test+"]");
    }

    after() : traceMethods()
    {
        Signature sig = thisJoinPointStaticPart.getSignature();
        log.debug("Leaving:␣[" + sig.getDeclaringType().getName()+"."+sig.getName()+"␣test:␣"
            +test+"]");
    }
}
```

Listing 9: Loggingaspect

Dieser Aspect gibt für jeden Methodenaufruf im definierten Fokus je eine Zeile für den Beginn der Methode und das Beenden der Methode aus. Die Ausgabe enthält den Methodennamen sowie den Klassennamen. Daneben wird ein Zähler jeweils erhöht. Dieser zählt durchgängig von Null aufwärts, da ein Aspect standardmässig ein Singleton ist.

3.2.2 Sicherheit für eine einzelne Anwendung

Hier soll gezeigt werden, wie durch den Einsatz von AOP die Forderung nach einem Sicherheitskern (vgl. [Eck01, Seite 122]) auf elegante Weise realisierbar ist.

```
import javax.security.auth.Subject;
import javax.security.auth.login.*;
import com.sun.security.auth.callback.TextCallbackHandler;

public aspect Authentication {
```

```
    private Subject _authenticatedSubject;

    public pointcut authOperations()
        : execution(public * *.main(String[]));

    before() : authOperations() {
        if(_authenticatedSubject != null) {
            return;
        }
        try {
            authenticate();
        } catch (LoginException ex) {
            throw new AuthenticationException(ex);
        }
    }

    private void authenticate() throws LoginException {
        LoginContext lc = new LoginContext("Sample",
                                           new TextCallbackHandler());
        lc.login();
        _authenticatedSubject = lc.getSubject();
    }

    public static class AuthenticationException
        extends RuntimeException {
        public AuthenticationException(Exception cause) {
            super(cause);
        }
    }
}
```

Listing 10: authentication Aspect

Bevor die main-Methode des veränderten Programms ausgeführt werden kann, wird kontrolliert, ob der Benutzer sich authentifiziert hat. Falls dem nicht so ist, so wird versucht, eine Authentifikation zu erreichen. Wenn die Authentifikation fehlschlägt, so wird ein Fehler erzeugt und weitergereicht. Dies wird durch den before()-Advice beschrieben.

3.2.3 Policy Enforcement

„Policy Enforcement" ist ein Konzept zur Durchsetzung von Regeln innerhalb eines Projektes. Richtlinien können hiermit bei der Programmentwicklung nicht nur schriftlich oder mündlich den Entwicklern mitgeteilt werden, sondern in Regeln gefasst und durch Aspecte formuliert werden. Diese Regeln werden dann durch den Compiler überwacht und durch Compile-Time Declarations (siehe 3.1.1) beschrieben.

Eine sehr einfache Regel könnte zum Beispiel sein, dass der `log`-Befehl aus Performencegründen nicht mehr verwendet werden soll. Der entsprechende Aspect wird in Listing 11 gezeigt.

```
import java.util.logging.*;

public aspect DetectLogUsage {
    declare warning : call(void Logger.log(..))
        : "Consider Logger.logp() instead";
}
```

Listing 11: "Keine log Befehle" Aspect

Dieser Aspect hat zur Folge, dass bei der Compilation des Systems für jede Verwendung des `log`-Befehls eine Warnung erzeugt wird. In [Lad03] werden weitere Beispiele hierfür vorgestellt.

3.2.4 Exception Softening

Ein weiteres Konzept ist das „Exception Softening”. Java unterscheidet zwei Kategorien von Fehlern. Bei einer „checked Exception” wird bereits zur Compilation kontrolliert, ob dieser Fehler abgefangen oder explizit über den `throw`-Befehl weitergereicht wird. Eine „unchecked Exception” hingegen kann nicht daraufhin überprüft werden. Sie ist von Typ `RuntimeException` oder `Error` und wird nicht explizit überprüft durch den Compiler. Exception Softening erlaubt es Exceptions zu verwenden, ohne diese in der Methode, in der der Fehler entsteht, behandeln zu müssen.

3.3 Verwendung in Ant

Ein automatisches Weaven innerhalb eines Ant-Build-Prozesses ist möglich, indem der normale Java-Compiler durch durch den AspectJ eigenen Compiler ausgetauscht wird. Dies erfordert lediglich wenige Zeilen in der build.xml die in Listing 12 gezeigt werden. Nun kann als Compilertag auch iajc verwendet werden, welches den AspctJ-Compiler (ajc) als Compiler aufruft.

```
<taskdef resource="org/aspectj/tools/ant/taskdefs/aspectjTaskdefs.properties">
  <classpath>
      <pathelement path ="../devlib/aspectjtools.jar"/>
  </classpath>
</taskdef>
```

Listing 12: Änderungen an der build.xml

3.4 Integration in Tomcat

Da ein durch AOP verändertes Programm weiterhin korrekter Java-Bytecode ist, muss für ein bereits compiliertes und verwobenes Programm keine Veränderung am Tomcat vorgenommen werden. Nun kann es aber gewünscht sein, dass das Weaving durch den Tomcat-Server durchgeführt wird. Dies bringt den Vorteil, dass die Aspecte im Betrieb ausgetauscht werden können. Hierzu muss der Compiler von Tomcat durch den von AspectJ in der Konfigurationsdatei web.xml ausgetauscht werden. Folgende Zeilen in der web.xml werden benötigt:

```
  <servlet>
      <servlet-name>ajsp</servlet-name>
      <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
      <init-param>
          <param-name>logVerbosityLevel</param-name>
          <param-value>DEBUG</param-value>
      </init-param>
      <init-param>
          <param-name>compiler</param-name>
          <param-value>org.aspectj.tools.ant.taskdefs.Ajc11CompilerAdapter</param-value>
      </init-param>
      <init-param>
          <param-name>fork</param-name>
          <param-value>FALSE</param-value>
      </init-param>
      <load-on-startup>2</load-on-startup>
  </servlet>
 <servlet-mapping>
      <servlet-name>ajsp</servlet-name>
      <url-pattern>*.jsp</url-pattern>
```

```
    </servlet-mapping>
```

Listing 13: Änderungen an der web.xml

Nun muss dem Compiler noch mitgeteilt werden, wo bzw. wie er potentielle Aspecte findet. Dazu bietet der ajc eine Schnittstelle an, die es ermöglicht, die Compilerparameter zu ändern. Um dies möglich zu machen, muss wie folgt eine Umgebungsvariable gesetzt werden:

```
export CATALINA_OPTS=-Dorg.aspectj.tools.ant.taskdefs.AjcTask.COMMAND_EDITOR=ajee.adapters.
    AutoAspectpathRewriter
```

Listing 14: Compilerparameter

Die hier angegebene Klasse durchsucht vor dem Compilationsprozess verschiedene Verzeichnisse nach binärvorhandenen Aspecten, um diese dann dem Compiler mitzugeben. Sie ist auf der zugehörigen CD zu finden. Leider reagiert dieses System nicht auf das Austauschen der Aspecte, da diese Erweiterung nur aktiv wird, wenn der Tomcat JSP-Dateien kompiliert. Um einen Austausch eines Aspects zu erreichen, ist es nötig, im work-Verzeichnis den Applikationspfad zu löschen. Dadurch wird der Tomcat gezwungen, neu zu kompilieren und die Aspecte zu verwenden. Da dieser Vorgang durch ein Ant-Skript elegant zu lösen ist, stellt er keine echte Einschränkung dar.

3.5 Vorteile und Probleme

Der Einsatz der aspektorientierten Programmierung im Rahmen der Entwicklung eines Sicherheitsmoduls bringt verschiedene Vorteile. In [WJP03] wird festgestellt, dass das Modularisieren von Crosscutting Concerns (siehe 3.1) die **Verständlichkeit** und **Analysierbarkeit** des Quelltextes stark erhöht. Die Entwickler müssen nicht den gesamten Quelltext untersuchen, sondern können sich auf die Entwicklung eines Moduls beschränken.

Weiter wird in [WJP03] das Problem der **Verifizierbarkeit** angesprochen. Die Qualitätssicherung ist ein wichtiger Bestandteil bei der Entwicklung von Software. Das Verwenden von AOP hat auf diesen Prozess positive wie auch negative Auswirkungen. Der starke Modularisierungsansatz erzeugt kleinere und abgegrenzte Einheiten. Dies hat zur Folge, dass es einfacher ist, für diese Module Testfälle zu definieren und sie gegen diese Tests zu verifizieren. Problematisch ist aber die Kontrolle des Fokus des Aspects. Die Frage, ob der Aspect wirklich überall hinzugefügt wird, wo er nötig ist und er wirklich nur hinzugefügt wird, wo er benötigt wird, ist leider bisher durch kein Tool beantwortbar. Eine nachträgliche Kontrolle, ob wirklich dieses Ziel erreicht wurde, fehlt komplett.

AOP-Techniken bedeuten aber auch für den Einsatz von Software neue Risiken. Durch die Möglichkeit, den Bytecode im Nachhinein zu ändern, wird es für den Benutzer schwerer zu beurteilen, ob die Software, die er einsetzt, wirklich so

von dem Hersteller kommt. Allerdings können hier die in Java bereits etablierten Mechanismen zur Signierung verwendet werden. Der Anwender bekommt auf diesem Weg die Sicherheit, dass das Programm seit der Signatur durch den Hersteller nicht verändert wurde. Als zweites Sicherheitskonzept ist die Javasandbox zu nennen, die im nächsten Kapitel behandelt wird.

Der Einsatz von AOP in J2EE Umgebungen haben P. Slowikowski und K. Zielinski in [SZ03] untersucht. Sie sind hierbei zu dem Schluss gekommen, dass es oft nicht möglich ist, mit den in J2EE vorgesehenen Methoden komplizierte oder dynamische Sicherheitsdefinitionen zu formulieren. Somit wird oft auf „Programmatic Security" zurückgegriffen, also Sicherheit, die durch den Programmierer im Sourcecode verankert wird. Hier bietet sich nach Ansicht der Autoren von [SZ03] ein wichtiger Einsatzzweck von AOP.

In [BGP04] stellen T. Bühren, V. Gruhn und D. Peters ein dem AOP Ansatz verwandtes Konzept vor. Hier werden Wrapper eingesetzt, die die vorhandenen Klassenbibliotheken kapseln. Dadurch sollen beliebige Klassen zur Laufzeit ausgetauscht werden. Aufgrund der besseren Normung von AOP (z.B. in Form von AspectJ) und der mächtigeren Sprachelemente ist ein Wrappen durch aspektorientierte Programmierung einfacher formulierbar und durchführbar.

3.6 Javasicherheit

Bei der Betrachtung von Sicherheit gilt es zwei Begriffe zu unterscheiden. „Safety" besagt, dass auch in Fehlerzuständen das Programm noch das „Richtige" macht und es die Möglichkeit besitzt, auf Fehler zu reagieren. Durch „Security" werden Zugriffskontrollmechanismen und Resourcenkontrolle beschrieben.

Java ist eine auf Sicherheit hin entwickelte Sprache, da sie im Internetumfeld ihre Anwendung findet. Java-Programme können reisen und auf verschiedenen Plattformen bei den Benutzern ausgeführt werden. Somit ist von Java eine höhere Sicherheit zu erwarten, als von anderen Sprachen die es auf dem Markt gibt.

Im Sprachdesign sind die Konzepte der **Memory Safety** und **Type Safety** berücksichtig worden. Memory Safety besagt, dass Java keine Speicherstrukturen unkontrolliert dem Benutzer gibt. Somit gibt es keine Pointerarithmetik oder Arrays die zu lang sind. Durch Type Safety wird beschrieben, dass durch den Javacompiler sicher gestellt wird, dass auf Methoden und Operatoren nur erlaubte Operationen ausgeführt werden. Eine weitere Besonderheit ist die Existenz eines **Runtime Environment**, das ebenfalls die oben beschriebenen Kriterien überwacht. Ein Programm wird in einer sogenannten Sandbox betrieben.

In [HM01] wird von Hartel und Moreau die Frage gestellt, wie weit man der Sicherheit von Java vertrauen kann. Sie stellen fest, dass neben der Type und Memory Safety zwei weitere Eigenschaften wichtig sind. Die Erste ist die Möglichkeit zur Beschränkung von Ressourcen, die einem Programm zur Verfügung stehen. Hier gibt es Ressourcen, die einfach zu beschränken sind wie z.B. der Zugriff auf Dateien und Ressourcen, die sich nur schwer begrenzen lassen, so wie die

Rechenzeit, die von einem Programm konsumiert wird. Die zweite Eigenschaft ist das Auditing also das Überwachen und forensische Prüfen, ob das System wirklich sicher ist aufgrund der Entscheidungen, die es getroffen hat. Hierzu ist den Autoren von [HM01] keine Literatur bekannt.

Bei der Java-Architektur gibt es verschiedene Teile, die betrachtet werden müssen, um über die Sicherheit der Sprache eine Aussage zu treffen. Die wichtigsten Teile sind der „Dynamic Class Loader", die „Byte Code Verification" und der „Security Manager".

Javaprogramme werden nicht wie z.B. bei C/C++ zu einer Datei gelinked, sondern es wird der **Dynamic Class Loader** verwendet, der zur Laufzeit dynamisch Klassen lädt. Um sicherzustellen, dass der richtige Code mit den richtigen Signaturen geladen wurde, werden verschiedene Tests durch den Byte Code Verifier durchgeführt.

Dieser **Byte Code Verifier** führt folgende Kontrollen an den Klassen zur Laufzeit durch:

- Over- und Underflows von Stack Frames
- Validität des Bytecodes
- Sprünge dürfen nur zu korrekten Anweisungen führen
- Methodensignaturen müssen valide Informationen tragen
- die Operationen werden auf den richtigen Typen ausgeführt
- Zugriffskontrollen
- Objektinitialisierung vor dem ersten Gebrauch
- Exceptions und synchronized Statements werden nach FIFO behandelt

Der **Security Manager** ist eine Laufzeitkomponente, in der kontrolliert wird, ob die von einem Programm verlangten Zugriffe auf Ressourcen gewährt werden oder nicht. So wird zum Beispiel im Rahmen des Security Managers auch der Stack kontrolliert. Innerhalb des „Security Manager" existiert der „Policy Manager", dessen Aufgabe es ist, die Menge der Rechte zu bestimmen, die ein Programm besitzt. Dieser Manager ist über eine bestimmbare Datei konfigurierbar.

Eine genaue Sicherheitsdiskussion findet sich z.B. in [HM01] und [BBF02].

Trotz dieses hohen Sicherheitsniveaus beschreibt z.B. Marc Schönefeld in [Sch04] einen möglichen Angriff auf dieses Modell. Hier ist es nötig, dass das Java Runtime Environment dem Angreifer zugängig ist. Bei einem J2EE Applicationserver ist dies aber nicht der Fall. Die gesamte Kommunikation mit diesem Server erfolgt über Strings, die zwischen den Clients und dem Server ausgetauscht

werden. Ein Angriff auf diesen Server muss also einen Buffer Overflow oder ähnliches ausnutzen, falls es keinen anderen Weg gibt, den Server zu kompromittieren. Eine genauere Betrachtung dieser Problematik findet sich in Kapitel 6.

Java bietet also eine ausgereifte und stabile Basis für die Entwicklung sowie den Betrieb von webbasierten Diensten. Besonders die implementierten Eigenschaften der Sandbox sprechen stark für Java und gegen andere Sprachen wie zum Beispiel C. Durch den Einsatz von aspektorientierter Programmierung ist es möglich, das Design eines Systems stark zu vereinfachen. Dies kann die Stabilität und Sicherheit eines Systems entscheidend verbessern.

4 Referenzmonitor

Für die Implementierung der Zugangskontrollen wird das Konzept eines Referenzmonitors gewählt. Durch diesen wird ein vom Authentifikations- und Autorisationsverfahren unabhängiges Konzept beschrieben, wie diese in einem System verwendet werden können. Das Konzept des Referenzmonitors wird im Anschluss weiter erläutert. Danach wird im zweiten Teil RBAC als verwendete Sicherheitsstrategie eingeführt und im dritten Teil schrittweise um das Konzept der Workflows erweitert.

Ein Referenzmonitor wird im Federal Standard 1037C [fTSI96] wie folgt definiert:

> **reference monitor:** [An] access control concept that refers to an abstract machine that mediates all accesses to objects by subjects. [NIS]

Entsprechend dieser Definition ist ein Referenzmonitor eine Blackbox, welche gefragt wird, ob ein spezieller Benutzer das Recht hat, eine von ihm angeforderte Operation durchzuführen. Somit stellt der Referenzmonitor das Modul dar, in dem die Sicherheitsstrategie verankert ist.

Der Monitor kann in zwei getrennte Probleme unterteilt werden:

- Technische Implementierung des Referenzmonitors
- Wahl der richtigen Sicherheitsstrategie

4.1 Implementierung eines Referenzmonitors

Wie Anderson 1972 in [And72] beschreibt, sind die folgenden Punkte wichtige Charakteristika eines Referenzmonitors. Dieser soll

- manipulationssicher,
- so klein wie möglich sein, so dass er durch Tests oder Analyse auf korrektes Verhalten getestet werden kann und
- durch jede sicherheitsrelevante Stelle des Programmablaufs aufgerufen werden können.

Aus diesen Forderungen an die Software entwickelten sich verschiedene Ansätze, Sicherheit zu gewährleisten.

Der erste Ansatz bestand in der Erweiterung bestehender Betriebssysteme wie es in Abbildung 13 dargestellt wird. In diesem Modell stehen nur eine begrenzte Anzahl von Programmereignissen zur Verfügung, die durch das Betriebssystem kontrolliert werden können. Insbesondere

- die Ausführung spezieller Opcodes, welche einen Kontextwechsel zum Betriebssystem zur Folge haben oder
- die Überwachung der Speicherzugriffe durch eine spezielle Erweiterung der Hardware.

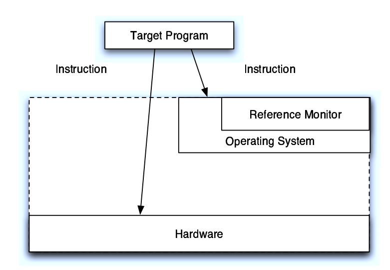


Abbildung 13: Referenzmonitor als OS Bestandtteil

Eine andere Möglichkeit ist die Verwendung eines Interpreters, in dem der Referenzmonitor integriert ist. Hier gibt es im Gegensatz zur erstgenannten Möglichkeit keine direkten Zugriffe auf die Hardware. Abbildung 14 skizziert das Konzept, das aber bisher keine so starke Verbreitung fand, da es eine starke Reduzierung der Ausführungsgeschwindigkeit zur Folge hat. Aktuelle Entwicklungen im Rahmen von Java und .Net verwenden allerdings diesen Weg.

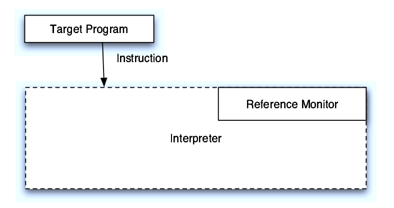


Abbildung 14: Referenzmonitor im Interpreter integriert

Wie Fred Schneider in [Sch03] feststellt, ist für die Implementierung eines Referenzmonitors der Ansatz eines Program Rewriter (Abb. 15) sehr erfolgsversprechend. Dieser modifiziert jedes Objekt bevor es zur Ausführung kommt. Aufgrund dieses Ansatzes werden Tests der Ausführungsrechte auf Basis der Methoden, der übergebenen Parameter und der jeweiligen Ausführungsrechte des Benutzers ermöglicht. So kann auf einfache Art eine feinkörnige Zugriffskontrolle im Nachhinein integriert werden.

4.2 Sicherheitsstrategie

1975 veröffentlichten Jerome Saltzer und Michael Schroeder folgende Idee in [SS75]:

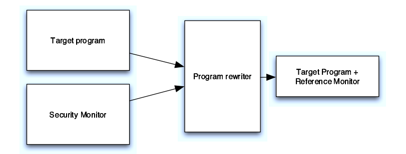


Abbildung 15: Inline Referenzmonitor

> Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job. [...]

Weiter führen sie aus, dass durch das Prinzip der minimalen Rechte zwei verschiedene Ziele erreicht werden. Zum Einen reduzieren sich die Schäden die durch einen Unfall oder Fehler passieren können. Zum Anderen wird die Menge der potenziellen Interaktionen zwischen Programmen auf das Minimum reduziert, welches nötig ist, die gewünschten Fähigkeiten des Systems zu erzielen. Dadurch wird die Anzahl der zu überwachenden Programme minimiert. Weiter schreiben J. Saltzer und M. Schroeder:

> [...] if a mechanism can provide 'firewalls', the principle of least privileges provides a rationale for where to install the firewalls.

Unter Berücksichtigung dieses Grundsatzes wurden verschiedene Sicherheitsmodelle entwickelt. Die bekanntesten sind das (hierarchisch-)rollenbasierte Modell (RBAC), das Chinese-Wall Modell und Zugriffsmatrix-Modelle wie z.B. Bell-LaPadula.

Wir verwenden im Rahmen dieser Diplomarbeit das RBAC (Role Based Access Control) Modell. Dieses bietet verschiedene Vorteile gegenüber anderen Modellen. Er ist durch die Trennung der Rechte von den Benutzern wesentlich flexibler in seiner Anwendung und Rechte können feingranular vergeben werden. Ohne diese beiden Eigenschaften, Flexibilität und Feingranularität, besteht stets das Risiko das ein Benutzer mehr Rechte eingeräumt bekommt, als dies vorgesehen ist (vgl. [ANSI04]). RBAC wurde erstmals 1992 von D.F. Ferraiolo und D.R. Kuhn in [FK92] beschrieben. Die Zuordnung der Benutzer zu den Rollen geschieht durch einen Administrator. Eine Rolle wird in [FK92] wie folgt beschrieben:

> Roles are group oriented. For each role, a set of transactions allocated the role is maintained. A transaction can be thought of as a transformation procedure [1] (a program or portion of a program) plus a set of associated data items. In addition, each role has an associated set of individual members. As a result, RBACs provide a means of naming and describing many-to-many relationships between individuals and rights.

Somit unterscheidet sich RBAC von anderen Sicherheitsstrategien, da hier nicht die Subjekte im Mittelpunkt der Betrachtung stehen sondern die durchzuführenden Aufgaben (vgl. [Eck01, S. 129]).

In [San98] hebt R. Sandhu hervor, dass eine Rolle eine stabile Einheit innerhalb einer Organisation ist, die sich über die Zeit nur langsam ändert. Damit unterscheidet sich eine Rolle stark von einer einfachen Gruppierung. Eine Rolle repräsentiert alle Kompetenzen, die benötigt werden, eine spezielle Aufgabe zu erfüllen. Weiter stellt R. Sandhu fest, dass für ein rollenbasiertes System zwei Eigenschaften entscheidend sind:

- Es muss ungefähr gleich schwer sein, die Rollenmitgliedschaft und die Rollenrechte zu bestimmen.

- Die Kontrolle über die Rechte muss auf eine kleine Anzahl von Benutzern reduziert sein.

Bei all diesen Modellen stellt sich die Frage, wie der Referenzmonitor jeweils entscheidet, ob ein Zugriff gewährt werden darf oder nicht. Hier gibt es zwei grundsätzlich unterschiedliche Ansätze.

- Das System kann berechnen, ob ein Zugriff gestattet wird.

- Das System erhält diese Information von außerhalb.

Der erste Ansatz nimmt an, dass es möglich ist diese Entscheidung zu berechnen, was aber im allgemeinen nicht der Fall ist (siehe [Sch03]). Der Referenzmonitor ist auf eine externe Datenquelle angewiesen, in der die Informationen über Berechtigungen hinterlegt sind. Eine solche Datenbasis muss speziell gegen Veränderungen und unerwünschte Zugriffe geschützt werden. Daneben muss auch das Entstehen der Datenbasis kritisch betrachtet werden. Da die Entscheidungen nicht berechenbar sind, müssen sie durch einen Entwickler, Administrator oder ähnliche Person getroffen und in der Datenbasis hinterlegt werden. Für den Betreiber der Software, der im Allgemeinen keinen Einblick in die Funktion der Software hat, bedeutet dies, dass er einem externen Sicherheitsanbieter vertrauen muss.

Eine weitere wichtige Frage ist, wie das System auf Änderungen der Zugriffsrechte reagiert. Der Rechteentzug kann sofort propagiert werden, so dass der Benutzer ab dem Zeitpunkt des Rücknahme der Rechte diese nicht mehr verwenden kann. Es kann aber auch eine gewisse Zeit vergehen. Bei einem rein ticketbasierten System wie z.B. Kerberos, wird eine Änderung der Zugriffsrechte erst nach einigen Stunden aktiv, wenn der Benutzer versucht, ein neues Ticket zu bekommen. Durch diese Verzögerung wird es dem Benutzer möglich Operationen durchzuführen, obwohl ihm bereits die Rechte für diese genommen wurden.

4.3 Workflows

Die im vorherigen Abschnitt vorgestellten Sicherheitsstrategien benötigen eine Entscheidungsbasis, anhand derer sie beurteilen was ein Benutzer darf bzw. nicht darf. Der rudimentärste Ansatz besteht darin, dass der Benutzer nach erfolgreicher Authentifikation in der Applikation alles darf. Dies bedeutet, dass die Software speziell so entwickelt sein muss, dass es keine Funktionen geben darf, die eventuell durch einen Benutzer nicht verwendet werden dürfen. Bei späteren Änderungen am Sicherheitsmodell führt dieser Ansatz dazu, dass Menüs durch extra Passwörter oder ähnliches gesperrt werden.

Ein wesentlich eleganterer Ansatz besteht darin, den Benutzer in seinem Verhalten in der Software zu kontrollieren und zu lenken. Die Software wird als Universalsystem für alle anfallenden Aufgaben innerhalb der Organisation entwickelt und ein Benutzer bekommt nur bestimmte Bereiche innerhalb der Software zur Verwendung freigegeben. Im Vorfeld muss hierbei geklärt werden, welche Aufgaben welcher Mitarbeiter auszuführen hat. In diesem Zusammenhang wird von Arbeitsabläufen bzw. Workflows gesprochen. Wir verstehen unter einem Workflow die Reihenfolge der Bildschirmmasken, die zur Bearbeitung einer Aufgabe nötig sind sowie die Benutzerinteraktion, die nötig ist, um von Maske zu Maske zu gelangen. Eine Repräsentation dieses Zusammenhangs kann als endlicher Zustandsautomat formuliert werden. Jede Maske ist hier ein Zustand und die benötigten Benutzereingaben stellen die Kanten dar. Um dies zu verdeutlichen kann als Beispiel ein Logindialog genommen werden. Die Applikation soll einen Dialog besitzen, dessen Funktion es ist, einen Benutzer zuzulassen, wenn er sich BERT nennt und sonst einen Fehler anzuzeigen. Dies wird in Abbildung 16 beispielhaft als Automat dargestellt. Auf diese Art und Weise lassen sich auch komplexere

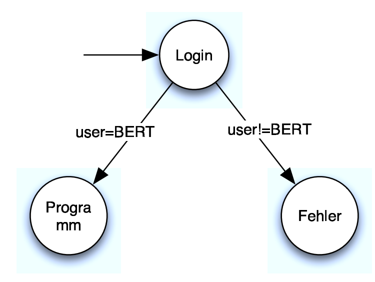


Abbildung 16: Beispiel eines Ablaufs

Abläufe erfassen.

Ein Referenzmonitor besitzt eine Menge von Workflows, die einzelnen Benutzern zugewiesen werden. Bei jedem Interaktionsschritt der Software vergleicht der Monitor die Benutzereingaben mit den erlaubten Werten, die in den Workflows hinterlegt sind. Auf dieser Basis kann nun entschieden werden, ob ein Schritt in der Interaktion abgelehnt werden muss oder ob er erlaubt ist.

Eine erste Erweiterung dieses Modells ist das Einführen von regulären Ausdrücken für die Überprüfung der Eingaben des Benutzers. Durch die Verwendung von regulären Ausdrücken, eine genauere Darstellung der Möglichkeiten findet man in [Fri97], können z.B. Wertebereiche definiert werden. Eine denkbare Anwendung hierfür sind Finanztransaktionen. Ein normaler Mitarbeiter darf nur 1000 Euro überweisen. Sein Vorgesetzter darf aber bis zu 10000 Euro transferieren. Der Term im ersten Fall würde so aussehen: „[1]+[0-9]{0,3}", im zweiten Fall:„[1]+[0-9]{0,4}".

Eine zweite Erweiterung der Workflows ist die Einbeziehung der Applikationsdatenbank. Wenn z.B. in einem Menü nur Hunde und Katzen zur Auswahl stehen kann dies durch eine Datenbankanfrage festgestellt werden und der Monitor erlaubt in diesem Fall auch nur die Angabe von „Hund" oder „Katze". Hierdurch können Datenbankzustände der Wirtsapplikation in die Entscheidungsfindung des Monitors einbezogen werden. Auch wird es möglich, Korrektheitsbedingungen der Informationen in einer Datenbank nachträglich zu definieren, ohne die Datenbankdefinition zu ändern.

4.4 Kombination aus Workflows und RBAC

Da in RBAC die durchzuführenden Aufgaben im Mittelpunkt der Betrachtung stehen, bietet sich eine Kombination mit Workflows an. Workflows sind gerade die Aufgaben, die ein Mitarbeiter im Rahmen seiner Tätigkeit erfüllt und lassen sich wie in Abbildung 17 dargestellt in das RBAC-Modell einfügen. In [KS02] zeigen

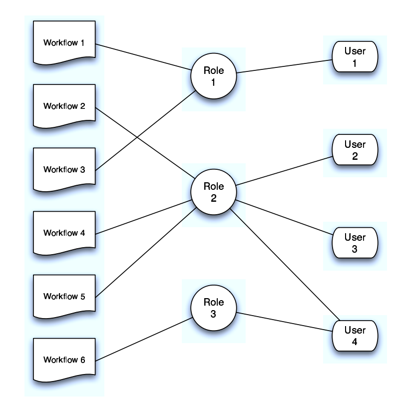


Abbildung 17: Einsatz von Workflows in RBAC

R. Sandhu und S. Kandala die Kombination von RBAC und Workflows. Die in dieser Diplomarbeit verwendete Kombination gestaltet sich etwas einfacher, da keine Hierarchie der Rollen, wie sie in RBAC96 [SCFY96] existiert, vorgesehen ist. Die Rechte interpretieren wir hier als das Recht, ob ein Workflow (und damit auch alle States des Workflows) ausgeführt werden darf oder nicht. Diese Rechte werden von Sandhu und Kandala als **explizite Rechte** bezeichnet. Da aber die

Entscheidung über die Korrektheit eines Schrittes innerhalb der Modellierung des Workflows getroffen wird, gibt es daneben noch die **impliziten Rechte**. In jedem Schritt wird versucht, den Workflow in einen neuen gültigen Zustand (State) zu bringen. In Abbildung 18 ist ein kleiner Ablauf gezeigt. Dort wird bereits in Betracht gezogen, dass das System später den eigentlichen Methodenaufruf kapselt, also „test rigths" als Zustand eingeführt wird. Formal beschrieben besitzt

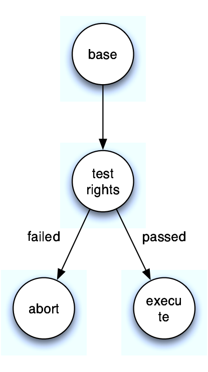


Abbildung 18: Beispielworkflow

das RBAC96 Modell folgende Komponenten:

U – Menge von Benutzern, R – Menge von Rollen, P – Menge von Rechten
$UA \subseteq U \times R$ (Zuordnung User – Rollen)
$RH \subseteq R \times R$ (partielle Ordnung auf den Rollen, $\leq$)
$PA \subseteq R \times P$ (Zuordnung Rollen – Rechte)
$permissions : R \rightarrow 2^P$ (jeder Rolle r wird eine Menge von Rechten zugeordnet)
$permissions* : R \rightarrow 2^P$ (erweiterte Relation um Rollenhirarchie)
$permissions(r_i) = \{p \in P \mid (p, r_i) \in PA\}$
$permissions(r_i)* = \{p \in P \mid (\exists r \leq r_i) \mid [(p, r) \in PA]\}$

In [KS02] wird diese grundlegende Definition in zwei Stufen erweitert, um Workflows und States verwenden zu können. In der ersten Erweiterung werden die sog. Tasks eingeführt, die den Workflows entsprechen. Zu diesen Tasks werden Operationen definiert. In dieser Erweiterung ist lediglich ein „Ausführen" als Operation sinnvoll. Neben den Tasks gibt es noch die Instanzen der Tasks sowie eine Abbildungsfunktion($\Im$), die den Zusammenhang zwischen den Tasks und den Instanzen herstellt. Das Modell, das sich aus diesen Voraussetzungen ergibt sieht wie folgt aus:

U, R, RH, UA werden unverändert aus der RBAC96 Definition übernommen.
OP = {execute}
TT – Menge von Tasks (Workflows), TI – Menge von Instanzen
$\Im : TT \rightarrow 2^{TI}$ such that $\Im(a) \cap \Im(b) = \emptyset$ and $a, b \in TT$
EP (explicit permissions) = $OP \times TT$
IP (implicit permissions) = $OP \times TI$
P (set of permissions) = $EP \cup IP$
EPA (explicit assigned permissions) $\subseteq R \times EP$
IPA (implicit assigned permissions derived from EPA)
IPA = $\{(r_i, execute, t_i) \mid [\exists(t_i, execute, t) \in EPA] \wedge t_i \in \Im(t)\}$
PA = $EPA \cup IPA$ (Permission Assignment)
$permissions : R \rightarrow 2^P$ (jeder Rolle r wird eine Menge von Rechten zugeordnet)
$permissions* : R \rightarrow 2^P$ (erweiterte Relation um Rollenhirarchie)
$permissions(r_i) = \{(execute, t_i) \mid (\exists[(r_i, execute, t) \in EPA] \wedge t_i \in \Im(t))\}$
$permissions(r_i)* = \{(execute, t_i) \mid (\exists r \leq r_i)[(r, execute, t) \in EPA] \wedge t_i \in \Im(t)\}$

R. Sandhu und S. Kandala stellen in fest [KS02], dass diese Definition noch den Nachteil hat, dass ein Task nicht notwendigerweise beendet wird. Somit besteht keine Beschränkung, wie oft ein Task ausgeführt wird. Um diesem Problem gerecht zu werden, führt Sandhu den Begriff der Zustände eines Tasks ein.

Für jeden Zustand innerhalb eines Tasks gibt es zwei mögliche Operationen: abort oder commit. Für den Startzustand gibt es jedoch nur eine Operation die möglich ist: execute. Dies führt den Task in den „Executing" Zustand, wie in Abbildung 19 dargestellt.

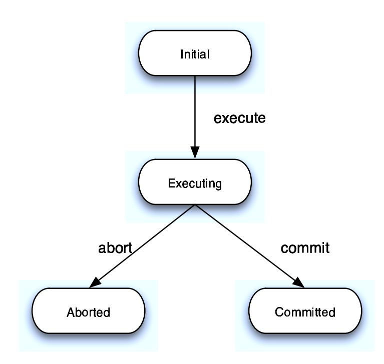


Abbildung 19: Task Struktur

Ein „abort" führt in einen Fehlerzustand und von dort wieder zurück in den Startzustand. „Commit" führt den Task in einen neuen gültigen Zustand. Um mit diesen Zuständen arbeiten zu können werden zwei Funktionen benötigt:

- **CurrentState**
  CurrentState liefert den aktuellen Zustand eines Tasks und wird wie folgt definiert:
  $CurrentState : TI \rightarrow S$ Ordnet jeder Taskinstanz Zustände zu
  $CurrentState(t_i) = \{s \mid s \in S \ and \ s \ is \ in \ the \ current \ state \ of \ ti\}$

- **PossibleOperations**
  PossibleOperations bestimmt die möglichen Operationen in einem gegebenen Zustand.
  $PossibleOperations \subseteq S \times OP$, in unserem Fall kann also geschrieben werden:
  $PossibleOperations = \{(Initial, Execute), (Executing, commit), (Executing, abort)\}$

Diese Überlegungen führen zu der zweiten Erweiterung des RBAC96-Modells, in dem RBAC96 um Workflows erweitert ist. Die Workflows besitzen Zustände, auf denen nur wenige Operationen (execute, commit, abort) erlaubt sind.

U, R, RH, UA, TT, TI, EP IP werden aus der ersten Erweiterung unverändert übernommen.
OP = {execute, commit, abort} Menge der Operationen
S = {Initial, Executing, Commited, Aborted} Menge der Zustände
T = {(Initial, execute, Executing), (Executing, commit, Commited), (Executing,abort,Aborted)} Menge der Übergänge
$CurrentState(t_i) = \{s \mid s \in S \ and \ s \ is \ in \ the \ current \ state \ of \ t_i\}$
$PossibleOperations = \{(Initial, Execute), (Executing, commit), (Executing, abort)\}$
$P = EP \cup IP$
$EPA \subseteq R \times EP$ (Menge der Explizit zugewiesenen Rechte)
$IPA = \{(r_i, op, t_i) \mid [\exists (r_i, op, t) \in EPA] \wedge [t_i \in \Im(t)] \wedge [(CurrentState(t_i), op) \in PossibleOperations]\}$
$PA = EPA \cup IPA$
$permissions : R \rightarrow 2^P$ (jeder Rolle r wird eine Menge von Rechten zugeordnet)
$permissions* : R \rightarrow 2^P$ (erweiterte Relation um Rollenhirarchie)
$permissions(r_i) = \{(op, t_i) \mid [\exists (r_i, op, t) \in EPA] \wedge [t_i \in \Im(t)] \wedge [(CurrentState(t_i), op) \in PossibleOperations]\}$
$permissions * (r_i) = \{(op, t_i) \mid [\exists r \leq r_i][(r, op, t) \in EPA] \wedge [t_i \in \Im(t)] \wedge [(CurrentState(t_i), op) \in PossibleOperations]\}$

Hierbei ist zu beachten, dass der Besitzer einer Instanz auch einen Zustandswechsel durchführen darf. Durch das System muss dies sichergestellt werden.

In der Implementierung (Kapitel 8.5) wird eine weitere Eigenschaft dieser Definition verwendet. Der Benutzer kann gleichzeitig in mehreren Workflows/Tasks sein. Da in der Definition von Mengen gesprochen wird, ist die Unabhängigkeit der Tasks erzwungen und ein paralleles Verwenden von zwei Workflows ist gestattet, ohne die Sicherheitseigenschaften des Modells zu verletzen.

5 Authentifikation im Webumfeld

Die Authentifikation eines Subjektes spielt im Webumfeld eine sehr wichtige Rolle da auf Basis der Authentifikationsdaten die Identifikation eines Benutzers vorgenommen wird sowie der Zugriff auf Resourcen gestattet wird. Authentifikation wird in [SS96] beschrieben als:

> Authentication establishes the identity of one party to another. Most commonly authentication establishes the identity of a user to some part of the system.

Der Begriff Authentifikation bedeutet also generell, dass ein Subjekt seine Identität gegenüber einem anderen Subjekt oder System nachweist. Bezogen auf Computer weist ein Benutzer seine Identität einem System eines Computers nach, also eine Benutzer-zu-Computer Authentifikation. Da traditionelle Authentifikationen basierend auf physischen Inspektionen eines Subjektes für den Einsatz der Benutzer-zu-Computer Authentifikation schlecht praktikabel sind [BM03], wird diese auf Basis der drei Kernelemente: Wissen, Besitz und Biometrie aufgebaut ([SS96], [Eck03, Kapitel 10]).

- **Wissen**
  Die Authentifikation auf Basis von Wissen wird in heutigen Systemen am häufigsten verwendet. In der Regel authentifiziert sich der Benutzer durch die Kenntnis eines geheimen Passwortes.

- **Besitz**
  Die Authentifikationsmethode durch Besitz fordert, dass der Benutzer im Besitz eines Gerätes (Tokens) ist. Hierbei wird eigentlich das Gerät und nicht der Benutzer authentifiziert. Meistens wird die besitzbasierte Authentifikationsmethode mittels Chipkarten gelöst [Eck03]. Auf der Chipkarte ist ein geheimer Schlüssel gespeichert, mit dem die Chipkarte sich mittels eines Challenge-Response Verfahren ausweisen kann.

- **Biometrie**
  Biometrische Verfahren zur Authentifikation basieren auf den Merkmalen eines Menschen, die ihn eindeutig identifizieren. Hierzu gehören physiologische oder verhaltenstypische Eigenschaften eines Menschens.

Im Webumfeld, vor allem dem E-Commerce, kommt der Authentifikation eines Benutzer noch eine weitere Bedeutung zu. Das Vertrauen eines Käufers in einen Verkäufer beruht zum großen Teil darauf, inwieweit er der Identität seines Verhandlungspartners vertrauen kann. Das Vertrauen ist generell größer, wenn der potenzielle Schaden geringer ist [FPHKH00], d.h. je mehr man der Authentizität eines Verhandlungspartner vertrauen kann desto geringer ist der zu erwartende

Schaden den man im Gegensatz von einer zweifelhaft authentifizierten Person erwarten könnte.

Zur Verbesserung einer Authentifikation können mehrere Methoden gleichzeitig verwendet werden. Ein Verfahren, bei dem sich der Benutzer mittels Passwort und Token bei einem System anmeldet und zusätzlich per biometrischen Daten gegenüber dem Token ausweisen muß würde sowohl ein Verfahren nach Wissen sein, da der Benutzer ein geheimes Passwort vorweisen muss, als auch nach Besitz, da er das Token besitzen muss als auch nach Biometrie, da er sich gegenüber dem Token ausweisen muss.

5.1 Tomcat

Im Normalfall wird eine innerhalb des Tomcat-Container laufenden Applikation nicht abgesichert. Erst durch Angabe von Sicherheitseinstellungen im deployment descriptor (web.xml) der Applikation wird Tomcat angehalten, eine Sicherheitsüberprüfung durchzuführen. Mögliche Einstellungen können dabei, wie in der Java Servlet Spezifikation ([Cow01, Kapitel SRV.12]) angegeben, zu:

- Authentifikation,
- Zugriffskontrolle,
- Datenintegrität und
- Vertraulichkeit

gemacht werden.

Das Sicherheitssystem von Tomcat ist Benutzer-Rollen basiert. Ein Benutzer kann zu mehreren Rollen gehören als auch eine Rolle mehrere Benutzer beinhalten. Die Benutzernamen und Rollen werden in den Konfigurationsdateien angegeben. Da die Benutzer und Rollen für alle Applikationen die innerhalb eines Tomcat-Containers laufen gelten, wird ein Single Sign-On System definiert [caf]. Zu einer Webapplikation können Sicherheitskriterien passend zu den Rollen und Benutzern festgelegt werden. Die Authentifikation sowie die Zuordnung eines Benutzer zu einer Rolle wird vom Tomcatsystem übernommen.

5.1.1 Authentifikation

In der Java Servlet Spezifikation [Cow01] ist angegeben, dass es 4 Möglichkeiten gibt, wie sich ein Benutzer authentifizieren kann. Diese sind:

- HTTP Basic Authentication
- HTTP Digest Authentication
- Form Based Authentication

- HTTPS Client Authentication

Das HTTP Basic Verfahren ist ein sehr einfaches Authentifikationsverfahren, das in der HTTP/1.0 Spezifikation [BLFF] definiert ist. Der Web-Browser des Benutzers fragt nach Aufforderung des Servers den Benutzer nach Benutzernamen und Passwort. Das Ergebnis wird dann vom Browser im Klartext an den Server geschickt. Diese Art der Authentifikation ist nicht sehr sicher. Man könnte zwar durch die Benutzung einer Kanalabsicherung, wie HTTPS, verhindern das die Antwort im Klartext übertragen wird allerdings werden Benutzernamen und Passwort vom Browser automatisch bei jeder folgenden Anfrage erneut gesendet, sodass bei einem Wechsel zurück auf das normale HTTP-Protokoll nach der Authentifikation das Passwort wieder im Klartext übertragen wird.

HTTP Digest authentifiziert den Benutzer auf die gleiche Weise wie das HTTP Basic Verfahren, nur dass im Unterschied das Ergebnis nicht im Klartext sondern verschlüsselt übertragen wird. Allerdings ist es nicht sehr weit verbreitet. Die Tatsache, dass Benutzernamen und Passwort genauso wie beim HTTP Basic Verfahren bis zu einem expliziten Beenden des Browsers durch den Benutzer im Speicher bleiben, eröffnet Möglichkeiten, den Browser des Benutzers direkt anzugreifen.

Da man bei den beiden HTTP Authentifikationsverfahren das Aussehen der Passworteingabe der Browsers nicht verändern kann, wurde das Form Based Authentifikationsverfahren spezifiziert bei dem der Entwickler die Möglichkeit hat, die Login-Seite nach seinen Vorstellungen zu gestalten. Die Angabe der Login-Seite wird dabei im deployment descriptor angegeben und die Namen der Formular-Loginfelder sind vorgegeben. Bei einem Zugriff auf eine Webseite durch einen Benutzer überprüft Tomcat, ob sich der Benutzer schon authentifiziert hat und leitet ihn im negativen Fall auf die im deployment descriptor angegebene Login-Seite. Das Ergebnis der Login-Seite wird daraufhin vom Tomcat überprüft und dem Benutzer wird Zugriff auf die Applikation gewährt oder er wird auf eine Fehlerseite weitergeleitet. Dieses Verfahren hat genau wie das HTTP Basic Verfahren den Nachteil, dass Benutzernamen und Passwort im Klartext übertragen werden. Dies könnte durch Kanalverschlüsselung wie HTTPS verhindert werden. Diese Authentifikationmethode ist bei einem Wechsel zwischen HTTPS und HTTP den beiden vorherigen aus Sicherheitsaspekten überlegen, da Benutzernamen und Passwort nur einmalig übertragen werden.

Die vierte Möglichkeit der Authentifikation ist HTTPS Client Authentifikation. Hierbei wird die Authentifikationsmöglichkeit des SSL Protokolls verwendet. Es authentifizieren sich über Zertifikate sowohl der Server gegenüber dem Client als auch der Client gegenüber dem Server, sodass eine wechselseitige Authentifikation stattfindet. Der Benutzer muß für dieses Authentifikationsverfahren im Besitz eines Public Key Zertifikats sein. Der öffentliche Schlüssel dieses Zertifikats wird im deployment descriptor auf Seiten der Applikation angegeben. Da nur der Benutzer im Besitz des privaten Schlüssel ist, ist dieses Verfahren recht sicher.

Allerdings muss beachtet werden, dass sich nur der Rechner authentifiziert, auf dem das Zertifikat gespeichert ist. Somit ist es eventuell wünschenswert, dieses Verfahren zusätzlich mit einem der anderen zu kombinieren, um sicherzustellen, dass sich der richtige Benutzer an dem Rechner befindet.

5.1.2 Zugriffskontrolle

Die Zugriffkontrolle bei Tomcat bzw. einem application container wird von der Java Servlet Spezifikation [Cow01] unterschieden in deklarative Sicherheit und programmative Sicherheit.

- **Deklarative Sicherheit**
  Bei der deklarativen Sicherheit wird die Konfiguration der Zugriffskontrolle im deployment descriptor vorgenommen. Dies hat den Vorteil, dass die Sicherheitsstruktur einer Applikation außerhalb der eigentlichen Applikation spezifiziert wird. Im deployment descriptor werden die nötigen Spezifikationen zu Rollen, Zugriffskontrolle und Authentifikation angegeben. Die Überprüfung der Sicherheitsangaben im laufenden Betrieb wird vom Tomcat direkt übernommen.

- **Programmative Sicherheit**
  Falls die Möglichkeiten der deklarativen Sicherheit nicht ausreichen, kann die programmatische Sicherheit zusätzlich verwendet werden. Hierbei kann innerhalb einer Applikation auf die Sicherheitskonfiguration zugegriffen werden. Die API vom Tomcat-Container bietet die Funktionen getRemoteUser, isUserInRole und getUserPrincipal. Über diese Funktionen hat der Programmierer die Möglichkeit eigene Sicherheitsfunktionen einzubauen. Diese betreffen dabei aber nur die Zugriffskontrolle, da die Authentifikation weiterhin vom Tomcatserver übernommen wird.

5.1.3 JAAS

Das Java Authorization and Authentication Service (JAAS) Framework [LGK+99] wurde von SUN 1999 spezifiziert. Dieses Framework ist eine Erweiterung der Authentifikations- und Autorisationsmöglichkeiten für Java. JAAS ist nicht nur auf den Tomcat beschränkt, sondern bietet die Möglichkeit sicher festzustellen, wer gerade Java Code ausführt, egal ob der Code in einer Applikation, einem Applet, einem Bean oder einem Servlet läuft [Sun04]. Mit diesem Framework ist es möglich, fast beliebige Authentifikationsmodule einzubauen, so z.B. Kerberos, JNDI, etc. Allerdings gibt es für die Anbindung des Liberty Protokolls noch keine Spezifikation für JAAS. SourceID bietet nur die Möglichkeit, das Liberty Protokoll direkt an eine Applikation anzubinden.

JAAS übernimmt die Authentifikation eines Benutzers bietet aber für die Zugriffskontrolle nur eingeschränkte Möglichkeiten. Dies ist zwar über den dekla-

rative Ansatz von JAAS möglich womit aber die Sicherheitsaspekte nicht von der eigentlichen Applikation getrennt sind.

5.2 Anforderungen

Das Single-Sign-On System vom Tomcat bezieht sich nur auf Applikationen, die innerhalb dieses einen Tomcatservers laufen. Da aber Applikationen auch über mehrere Server verteilt sein können, ist für solche dieses System nicht geeignet. Ein weiteres Problem ist die geringe Auswahl an Authentifikationverfahren beim Tomcat. Außer dem Authentifikationsverfahren per SSL sind alle anderen mit recht großen Sicherheitsproblemen behaftet. Falls eine Applikation eine andere Methode wünscht, kann auch hier das System vom Tomcatserver nicht verwendet werden. Weiterhin ist die Integration einer Single-Sign-On Funktionalität in eine Applikation wünschenswert, die vor allem in verteilten Applikationen wichtig ist. Ein Sicherheitssystem bezogen auf die Authentifikation für eine verteilte Applikation sollte somit unter anderem folgende Eigenschaften aufweisen:

- Unterstützung vieler Authentifikationsverfahren
- Integration eines Single-Sign-On Server
- Reauthentifikation
- Einfache Integration in eine Applikation

Im Folgenden wird kurz auf mögliche Authentifikationsverfahren im Webumfeld eingegangen und Reauthentifikation beschrieben. Zur Integration eines Single-Sign-On Servers sei auf das Kapitel 2 verwiesen.

5.2.1 Authentifikationsverfahren

Passwort Das Passwort-Verfahren ist das in der Praxis am häufigsten eingesetzte Verfahren[Eck03] und beruht auf der Basis von spezifischem Wissen. Die weite Verbreitung des Passwortverfahren ist dabei auch das größte Sicherheitsproblem dieses Verfahrens. Da ein Mensch sich nicht beliebig viele Passwörter merken kann, wird häufig dazu übergegangen, das gleiche Geheimnis für verschiedene Portale zu wählen oder auf einen Zettel zu schreiben, der sich in der Nähe des Computers befindet. Diese Tatsache birgt natürlich große Sicherheitsprobleme, da man durch das Knacken eines Passwortes auf mehrere Portale Zugriff bekommt bzw. das Geheimnis auf dem Zettel kein richtiges Geheimnis mehr ist. Zusätzlich gibt es mit Passwörtern noch das Problem, dass sie abgehört und geraten werden können. Durch die Tatsache, dass sich das Geheimnis über längere Zeit nicht ändert, da ständige Veränderungen die Akzeptanz des Verfahrens bei den Benutzern erheblich senken würden, besteht weiterhin die Gefahr, dass

ein Angreifer durch das einmalige Abhören erfolgreiche Maskierungsangriffe tätigen kann. Dieses Problem kann durch die Einführung von einmal benutzbaren Passworten verhindert werden. Ein in der Praxis häufig eingesetztes Verfahren ist das S/Key-Verfahren [Hal94]. Da Benutzer sich meistens einfach zu merkende Geheimnisse wählen, hat ein „Password Cracking" Angriff unter Zuhilfenahme von Wörterbüchern große Erfolge. Es ist sinnvoll, sichere Passwörter mit großer Entropie, also einem großen Wert für die Zufälligkeit des Passwortes, zu wählen. Dies wiederum erhöht die Schwierigkeit für den Nutzer sich das Geheimnis zu merken. Die Unsicherheit von Passwörtern kann durch ein Passwort Managment zwar minimiert werden, bei dem der Benutzer regelmäßig sein Passwort ändern muss und nur gute Passwörter im Sinne einer hohen Entropie erlaubt sind. Da bei diesem Verfahren aber trotzdem großes Vertrauen in den Benutzer gelegt wird, muss ein Passwort Managment immer auch eine Schulung der Benutzer beinhalten.

Symmetrische Kryptosysteme basieren auf der Tatsache, dass sowohl der Claimant, der sich authentifizieren lassen will, als auch der Verifier, der die Authentifikation überprüft, im Besitz des gleiches Schlüssels sind und beide den gleichen Verschlüsselungsalgorithmus verwenden. (Definition siehe ISO Norm 9798 [97997]). Dieser geheime Schlüssel kann aus dem Passwort des Benutzers abgeleitet sein oder aber er ist zu komplex, als dass ihn sich ein Benutzer merken könnte, sodass er auf einer Chipkarte oder einem Rechner gespeichert wird. Die Überprüfung des Schlüssels wird dabei überwiegend mittels Challenge-Response-Verfahren getätigt Näheres zum Challenge-Response-Verfahren findet man in [Eck03, Kapitel 10.2.3]. Der Verifier schickt dem Claimaint eine Challenge, meistens eine Zufallszahl, die der Claimant mit dem Schlüssel verschlüsselt und das Ergebnis, die Response, an den Verifier zurückschickt. Der Vorteil ist, dass bei jeder Authentifikation eine andere Challenge übertragen wird und es somit einem Angreifer erschwert wird, einen Maskierungsangriff durchzuführen. Allerdings sollte hier eine wechselseitige Authentifikation stattfinden, da sich sonst ein Angreifer als Verifier maskieren und somit eine known-plaintext Attacke auf den Claimant ausführen könnte. Außerdem muss beachtet werden, dass ein Angreifer die Verbindung zwischen Claimant und Verifier kontrollieren und die Response vom Claimant abfangen könnte, diesen außer Gefecht setzt und sich daraufhin mit der gestohlenen Response ausweisen könnte. Dies kann durch eine Erweiterung des Challenge-Response-Systems verhindert werden. Ein weiteres Problem bei symmetrischen Kryptosystemen ist, dass ein Server, der Verifier, der für die Authentifikation zuständig ist, alle Schlüssel der beteiligten claimants speichern muss. Dieser Server ist deshalb besonders gegen Angriffe abzusichern.

Asymmetrische Kryptosysteme, oder auch Public-Key-Kryptosysteme, basieren auf zwei Schlüsseln, einen Öffentlichen und einen geheimen Privaten. Diese

haben die Eigenschaft, dass alle Öffentlichen frei zur Verfügung gestellt sind und die Privaten geheim sind und man aus dem Öffentlichen nicht ohne sehr großen Aufwand auf den Privaten schließen kann. So gibt es nicht wie bei der symmetrischen Kryptographie ein gemeinsames Geheimnis, das auf beiden Seiten geschützt werden muss und die geheimen privaten Schlüssel müssen auch nie übertragen werden. Außerdem ist es nur dem Besitzer des privaten Schlüssels möglich, eine mit dem öffentlichen Schlüssel verschlüsselte Nachricht zu entschlüsseln. Weiterhin kann auch ein Signieren einer Nachricht stattfinden, indem eine Nachricht mit dem privaten Schlüssel verschlüsselt wird und jeder, der im Besitz des passenden öffentlichen Schlüssels ist kann die Nachricht entschlüsseln und weiß, dass nur derjenige im Besitz des privaten Schlüssels die Nachricht verschlüsseln konnte. Die Authentifikation bei einem asymmetrischen Kryptosystem läuft dabei so ab, dass der Verifier dem Claimant eine Zufallszahl schickt und der Claimant diese Zufallzahl mit seinem privatem Schlüssel verschlüsselt und diese daraus entstandene digitale Signatur zum Verifier als Antwort schickt. Wichtig bei diesem Verfahren ist, dass der Verifier dem öffentlichen Schlüssel des Claimants vertrauen kann. Um dies überprüfen zu können, muss es eine Public-Key Infrastruktur geben. Näheres hierzu in [Eck03, Kapitel 7.6.1]. Ein Beispiel eines asymmetrischen Kryptosystem ist das Authentifikationsverfahren des SSL Protokolls.

Chipkarten Die Authentifikation mit Hilfe einer Chipkarte ist eine Form der besitzbasierten Authentifikation. Nur jemand, der im Besitz dieser Karte ist kann sich authentifizieren. Hierbei wird normalerweise die Chipkarte über ein Kartenlesegerät an den Computer angeschlossen und der Computer authentifiziert sich mit Hilfe des Schlüssels auf der Karte gegenüber dem Verifier oder der Schlüssel bleibt auf der Karte und diese tätigt die Authentifikation direkt.

Man unterscheidet drei Arten von Chipkarten [Eck03]. Einfache preiswerte Speicherkarten, die nur zur reinen Speicherung von Daten ausreichen. Intelligente Speicherkarten, die zusätzlich eine Sicherheitslogik meist zur PIN Überprüfung besitzen. Und drittens die Smartcards, die mit einem eigenen Mikroprozessor und programmierbarem Speicher ausgerüstet sind. Die Authentifikation mit Hilfe von Chipkarten basiert darauf, dass ein geheimer Schlüssel auf ihnen gespeichert ist. Dieser geheime Schlüssel wird entweder, wie bei den einfachen Speicherkarten, direkt zu dem Computer übertragen oder bleibt, wie bei der Smartcard, auf der Chipkarte und die Chipkarte besitzt ein eigenes Kryptosystem. Eine Authentifikation mit Chipkarten kann dabei sowohl mit einem asymmetrischen als auch mit einem symmetrischen Kryptoverfahren stattfinden.

Biometrie Eine Authentifikation auf Basis von biometrischen Merkmalen basiert auf charakteristischen physischen Eigenschaften oder besonderem Verhalten einer Person. Hierzu müssen erst in einer Lernphase Referenzwerte von der Person aufgenommen werden und personenspezifische Eigenschaften isoliert werden.

Der Vorteil einer biometrischen Authentifikation liegt darin, dass eine Person zur Authentifikation nichts außer sich selbst braucht. Daraus ergibt sich, dass man keine wichtigen Daten zur Authentifikation vergessen oder verlieren kann. Das Problem an der biometrischen Authentifikation ist der zuverlässige Vergleich der aktuell aufgenommenen Daten mit den gespeicherten Referenzdaten. Studien der Wirksamkeit von biometrischen Erkennungssystemen gibt es vom BSI unter [fSidI03a]

5.2.2 Reauthentifikation

Eine Authentifikation wird typischerweise einmal am Anfang einer Sitzung initiiert. Mit Reauthentifikationen, also einer wiederholten Authentifikation nach der initialen, versucht man nachzuweisen, dass der aktuelle Benutzer mit dem initial authentifizierten User identisch ist [PB04]. Zusätzlich kann man hiermit fortlaufend das Verhalten eines Benutzers überwachen. Für sicherheitsrelevate Aktionen ist ein wiederholter Nachweis der Identität eines Benutzers wünschenswert ([Eck03, Kapitel 4.8]). Ohne Reauthentifikationen ist das System möglichen Angriffen von Insidern ausgeliefert. Es könnte z.B. ein Angreifer Zugang zu einem Rechner gelangen der initial authentifiziert wurde und für kurze Zeit unbewacht ist aber nicht ausgeloggt wurde. Weiterhin gibt das einmalige Klauen eines Passwortes einem Angreifer die Möglichkeit sehr lange auf das System Zugriff zu bekommen, ohne dass es bemerkt wird.

Viren, Hackern und ähnlichen Angriffen von außerhalb wird mittlerweile eine große Bedeutung zugeordnet und Systeme werden dementsprechend auch immer besser abgesichert. Die Angriffe, die durch eine fälschlicherweiser vertrauten Person innerhalb eines Computersystems durchgeführt werden können sind dagegen zu wenig beachtet. Der Schaden dagegen, den eine solche Person duch den Diebstahl von geheimen Daten verrichten kann, ist nach dem Computer Security Survey [R.W03b] die teuerste Form von Computerverbrechen im Jahr 2003.

Zero interaction Authentication Zur Reauthentifikation ist es schlecht, ein Verfahren zu verwenden, das ständig die Interaktion des Benutzers fordert. Ein solches Verfahren ist sehr aufdringlich und ermutigt den Benutzer, das Verfahren abzuschalten oder zu umgehen [CN02]. Eine Zero-Interaction Authentication (ZIO) ist ein Authentifikationsverfahren, das keine Interaktion mit dem Benutzer benötigt. Ein Benutzer ist im Besitz eines Tokens, mit dem er sich regelmäßig reauthentifiziert. Das initiale Authentifikationsverfahren kann dabei eine Interaktion mit dem Benutzer fordern, aber alle weiteren Authentifizierungen mittels ZIO benötigen keine. In einem solchen Szenario wird dabei eine Authentifikation nach Wissen und Besitz durchgeführt. Zuerst die initiale Authentifikation nach Wissen zur Benutzerreauthentifikation und alle folgenden in der Sitzung zur Computerauthentifikation nach Besitz.

5.3 Wibu-Key

Mit Hilfe des Liberty Protokolls (siehe Kapitel 2.2) wird es einer Web-Applikation ermöglicht, den Prozess der Authentifikation komplett auszulagern. Der Single-Sign-On Server, im Kontext von Liberty der Identity Provider IDP, übernimmt die Aufgabe der Authentifikation für die Web-Applikation. Das Liberty Protokoll ist dabei so offen gehalten, dass es beliebige Authentifikationsverfahren unterstützt. Der Identity Provider kann je nach Authentifikationsverfahren spezielle Prozeduren mit dem Benutzer bzw. Clientrechner durchführen. Dieser teilt der Web-Applikation, dem Service Provider, nach erfolgreichem Ablauf des Authentifikationsverfahren mit, mit welchem Verfahren sich der Benutzer authentifiziert hat.

Wir entschieden uns in unserem Demonstrator für eine initiale Passwortauthentifikation, um ein Verfahren nach Wissen zu integrieren und zusätzlich zu einer möglichen Reauthentifikation durch ein Verfahren nach Besitz. Dies sollte ein Token sein, dass möglichst einfach an vielen Rechnern mit Internetanschluss anschließbar ist. Um eine einfache Integration in einem Tomcat basierten Identity Provider zu gewähleisten, muss eine Java-Schnittstelle zu diesem Token zur Verfügung stehen. Eine der wenigen Tokens, die diese Anforderungen erfüllen, ist der Wibu-Key [AG] der Firma Wibu Systems. Der Wibu-Key wird hauptsächlich als Kopierschutzsystem für Software eingesetzt, kann aber auch als Authentifikationssystem dienen [AG01] (nach A. Schmidt auch beim BMWA im Einsatz).

Details Der Wibu-Key beherrscht eine symmetrische Verschlüsselung nach dem FEAL Algorithmus mit 64-Bit-Schlüsseln. Den Wibu-Key gibt es nicht nur als USB-Version sondern auch als COM, LPT, PCMCIA, etc. Somit ist der Wibu-Key an fast jeden Computer anschließbar. Der Nachfolger des Wibu-Key vom Hause Wibu Systems - der Codemeter - hat wesentlich bessere Verschlüsselungskapazitäten als der Wibu-Key, allerdings gab es für den Codemeter Mitte 2004 noch keine Java-API, sodass er für unseren Demonstrator nicht in Frage kam.

Der Firm-Code und User-Code sind die wichtigsten Einträge im Wibu-Key. Diese werden über ein Programmiertool fest in den Wibu-Key eingespeichert. Der Firm-Code ist eine 24-Bit-Zahl, die jedem Kunden von Wibu Systems zugeteilt wird. Es wird dabei garantiert, dass jeder Firm-Code nur einmal vergeben wird [AG04]. Dieser Firm-Code kann nur mittels einer Firm-Security-Box in einen Wibu-Key programmiert werden. Somit ist die Firma Wibu System der Sicherheitsanker und repräsentiert quasi eine Zertifizierungsstelle, da nur sie die Möglichkeit besitzt, Firm-Codes zu vergeben beziehungsweise Programmierboxen für einen speziellen Firm-Code herauszugeben. Der User-Code kann mit einer Firm-Security-Box frei definiert werden. Der User-Code ist dabei genauso wie der Firm-Code eine 24-Bit-Zahl. Somit hat man mit einer Firm-Security-Box die Möglichkeit, jedem Benutzer einen eigenen User-Code zuzuteilen. Um den Wibu-Key

als Authentifikationswerkzeug verwenden zu können, werden durch die symmetrische Verschlüsselung zwei Wibu-Keys benötigt. Sowohl auf Seiten des Claimant als auch auf Seiten des Verifier. Allerdings bietet ein normaler Wibu-Key nur die Möglichkeit, 10 Einträge einzuspeichern, sodass mit einem Wibu-Key auf Seiten des Servers nur maximal 10 Benutzer authentifizierbar sind. Erst die Nachfolgeversion CodeMeter des Wibu-Keys mit der Möglichkeit über 1000 Einträge zu speichern ermöglicht eine bessere Skalierbarkeit.

FEAL Der FEAL Algorithmus [AS87], der im Wibu-Key verwendet wird, wurde ursprünglich von Shimizu und Miyaguchi entwickelt. FEAL ist ein Blockchiffre Algorithmus mit 64-Bit Schlüssellänge und sollte eine Alternative zum DES sein. Eines der Designziele war ein großer Geschwindigkeitsgewinn bei Softwareverschlüsselungen gegenüber DES. Der FEAL Algorithmus war ursprünglich auf 4 Runden, deshalb auch FEAL-4, ausgelegt, wurde aber recht schnell auf 8 Runden verdoppelt [MSS88] da es einige sehr leichte Angriffsmöglichkeiten auf den FEAL-4 gab. Einer sogar nur mit maximal 20 gewählten Klartexten [Mur90]. Allerdings wurde auch der FEAL-8 mittels Kryptoanalysen [BS91] gebrochen. Im Jahre 1994 wurde ein Kryptoanalyse Attacke gegen FEAL-8 entwickelt die nur 2^{25} gewählte Klartexte benötigt [OA94]. Hiernach wurde eine FEAL-N Version des ursprünglichen FEAL Algorithmus entwickelt, der N Runden durchführt. Allerdings gibt es für FEAL-N bis zu 31 Runden auch Kryptoanalytische Attacken. Der FEAL Algorithmus sollte wegen der Vielzahl von Angriffsmöglichkeiten als unsicher betrachtet werden [Lab00].

Der Verschlüsselungsalgorithmus des Wibu-Keys ist nach [AG04] - wie in Abbildung 20 dargestellt - eine Hintereinanderausführung mehrerer FEAL-8 Algorithmen. Zuerst wird der Firm-Code mit einer Konstanten als Schlüssel verschlüsselt. Das Ergebnis wird als Schlüssel für eine erneute FEAL Runde mit dem User-Code verwendet. Dieses Ergebnis wird daraufhin wieder als Schlüssel für eine FEAL Runde mit dem Selection-Code verwendet. Das Ergebnis hieraus bildet den Schlüssel für die eigentlichen Daten, die verschlüsselt werden sollen.

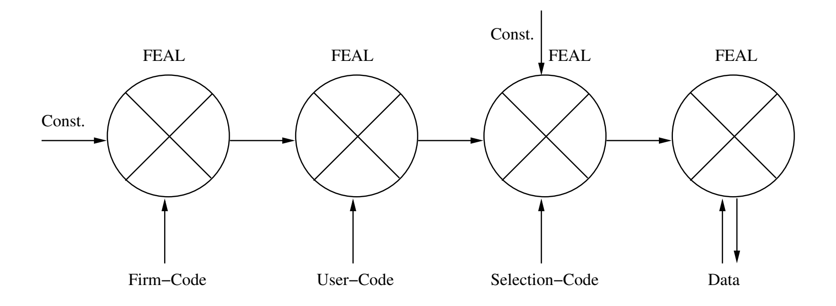


Abbildung 20: Wibu-Key Feal Algorithmus - [AG04]

Authentifikation Ein Authentifikationsverfahren mit dem Wibu-Key wird in [AG01] beschrieben. Hierbei handelt es sich um ein Challenge-Response-Verfahren. Bevor der Client Rechner Zugriff auf eine Webseite tätigen kann, schickt der Server (Verifier) dem Client (Claimant) eine Challenge, die der Client Rechner mit einer Response beantworten muss. Der Server schickt dem Client in der Challenge - oder auch Authentication Request Data genannt - einen zufälligen Selection Code und einen zufälligen String als Challenge. Der Client-Rechner verschlüsselt den zufälligen String mit Hilfe des Selection Codes als Schlüssel mit dem Wibu-Key. Das Ergebnis wird dann als Response zurück an den Server geschickt. Der Server muss jetzt seine Challenge selbst mit dem an den Server-Rechner angeschlossenen Wibu-Key verschlüsseln und die beiden Ergebnisse vergleichen. Falls sie identisch sind, ist der Benutzer authentifiziert.

API Die Interaktion mit der API vom Wibu-Key verläuft wie detailliert in [AG03] beschrieben. Zuerst muss mittels der Funktion WkbOpen2 der Zugriff auf den Wibu-Key initialisiert werden. Hierbei werden die Parameter Firm-Code und User-Code übergeben. Falls der Zugriff funktioniert können mittels der Funktion WkbCrypt2 Daten verschlüsselt werden. Dieser Funktion muss der Selection-Code sowie die zu verschlüsselnden Daten übergeben werden. Als Rückgabewert erhält man die mit dem Feal-Algorithmus verschlüsselten Daten bei denen der Firm-Code, User-Code sowie der Selection-Code als Schlüssel verwendet wurden (vgl. Abbildung 20). Zum Schließen einer Verbindung mit dem Wibu-Key wird die Funktion WkbClose2 aufgerufen.

6 Serverabsicherung

Um die Sicherheit eines gesamten Systems beurteilen zu können, müssen die verschiedenen Teile betrachtet werden, die zusammenarbeiten, um die Funktionalität zu erzeugen. Da hier ein webbasiertes System entwickelt wird, ist insbesondere die Möglichkeit von Angriffen zu untersuchen, die nicht nur auf die Applikation abzielen. Somit müssen neben dem Standort und der Hardware auch das Betriebssystem und die darauf aufsetzenden Dienste einer genauen Betrachtung unterzogen werden.

Wir beschränken uns hier auf eine kleine und in keiner Weise repräsentative Auswahl von Komponenten und Diensten mit dem Ziel, die Verwendung eines Applikationsservers sicher zu gestalten.

Sicherheit ist stets die Abwägung zwischen den Kosten und dem Nutzen der Sicherheitsmaßnahmen. Die perfekte Sicherheit ist nie zu erreichen, sondern immer nur eine angemessene. Um beurteilen zu können, ob ein System sicher genug ist, werden Metriken benötigt, anhand derer die Sicherheit quantifizierbar ist. Eine Auswahl wird im Kapitel 6.4 vorgestellt.

6.1 Hardware

Ein Glied der Sicherheitskette ist die Hardware und der Ort an dem diese steht. Bei Servern muss eine hohe Ausfallsicherheit erreicht werden, die durch Redundanzmaßnahmen erreicht werden kann. Dies können doppelte Netzgeräte sein, RAID Strategien bei den Festplatten (siehe [BDR86]) bis hin zu doppelten Servern, die eine failover-Strategie besitzen. Dies wird zum Beispiel vom Linux High-Availability Project [Pro85a] als Ziel verfolgt.
Der Standort des Servers sollte trocken sein und keine Wasserleitungen oder ähnliches in der Nähe haben. Daneben sollte auch an Brandschutzmaßnahmen, eine Notstromversorgung, physikalische Barrieren und ähnliches gedacht werden. Eine ausführliche Betrachtung der Standortproblematik findet sich im später vorgestellten Grundschutzhandbuch des BSI [fSidI03b].

Wichtig in einem Schadensfall ist es, dass eine klare Kompetenzenverteilung besteht, die sicherstellt, dass das System schnellst möglich wieder einsatzbereit ist.

Ein anderer wichtiger Punkt ist die Regelung des Zutritts zu den Servern. Ein direkter physikalischer Zugriff stellt eine große Gefahr für die Verfügbarkeit und Integrität des Dienstes dar. Zum Einen besteht die Gefahr der physikalischen Beschädigung bzw. Zerstörung der Hardware aber auch des Diebstahls der Hardware oder der Daten, die auf der Hardware gespeichert sind. Nicht vergessen werden darf die Manipulation von Hardware, Software und Daten. Aus dieser Betrachtung ergibt sich die Notwendigkeit einer Zutrittsbeschränkung zu den Räumen, in denen sich die Hardware befindet.

6.2 Sicherheitsmaßnahmen auf Betriebssystemebene

Die Maßnahmen, die auf Betriebssystemebene Anwendung finden, lassen sich in präventive Maßnahmen, Maßnahmen zur Erkennung von Angriffen und zur Wiederherstellung nach einem Angriff einteilen.

In den Bereich der präventiven Maßnahmen fallen alle Schritte, die eingeleitet werden, einen Angriff zu verhindern. Wichtig hierbei ist bereits die Wahl des Betriebssystems. In [JF03] wird empfohlen ein OpenBSD zu verwenden, da es eine geringe Häufigkeit an entdeckten Sicherheitslöchern in der Vergangenheit aufgewiesen hat. Dies stellt auch den Vorteil von offenen Betriebssystemen dar. Umso mehr Menschen in der Lage sind, die Fehlerfreiheit des Systems zu evaluieren, desto höher ist die Wahrscheinlichkeit, dass ein enthaltener Fehler auch gefunden wird. Nach der Wahl des Betriebssystems muss dieses für den geplanten Einsatz angepasst werden. Dies bedeutet, dass nur die absolut nötigen Dienste installiert sein dürfen und Dienste, die nur intern Verwendung finden, müssen durch eine entsprechende Konfiguration des Netzwerks nach aussen gesperrt werden. Hierzu werden hier nicht weiter behandelte Komponenten wie Router und Firewalls eingesetzt. Um den dann erreichten Sicherheitsstandard zu erhalten ist es nötig, das Betriebssystem und die darauf verwendeten Dienste stets aktuell zu halten und auf entdeckte Sicherheitslöcher schnell zu reagieren.

Wenn aber über eine nicht entdeckte Lücke ein Angriff stattfindet, sollte dieser ebenfalls erkannt werden. Hierfür finden verschiedene Ansätze Anwendung. Intrusion Detection Systeme (IDS) sollen Portscans und andere netzbezogene Angriffe erkennen und durch ein aktives Reagieren den Angriff unterbrechen. Wie in [VVK03], [IDS00] und [DCW+99] zu sehen ist, reicht aber die Existenz eines IDS-Systems an sich nicht aus. Eine sorgfältige Auswahl und Konfiguration ist notwendig. Bufferoverflow-Angriffe können durch Speicherschutzsysteme wie PAX (siehe [SPP+04]) und Filter (siehe [JF03]) in den Diensten begegnet werden. Falls ein Angriff Erfolg hatte, kann dies durch vergleichende Programme erkannt werden. Ein Vertreter dieser Klasse von Programmen ist Tripwire ([Tri04]), mit dessen Hilfe ausgewählte Dateien nach Veränderungen durchsucht werden können.

Um die Folgen eines Angriffs gering zu halten und die Ausfallzeiten zu minimieren, kommen redundante Systeme zum Einsatz. Unerlässlich sind Backups, die es ermöglichen, auf vorherige konsistente Zustände des Systems zurückzugreifen.

Ein funktionierendes Backup ist ein integraler Bestandteil jedes Schutzsystems, da durch dieses die Möglichkeit besteht, den Server wieder in einen funktionierenden Zustand zu versetzen. Daher ist bei einem Backup das Vollständigkeitsprinzip absolut wichtig. Auch muss die Aktualität und die Integrität des Backups sichergestellt werden. Um dies zu gewährleisten ist eine Datensicherungsstrategie erforderlich. In dieser muss auch geregelt sein, wie die Datenträger zu verwahren sind (vgl. [SLZ+04]).

6.3 Tomcats Sicherheitskonzept

Um ein J2EE-Programm ausführen zu können wird eine J2EE-konformer Applikationsserver benötigt, der die entsprechende Umgebung zur Verfügung stellt, die eine Webapplikation benötigt. Daher wird der Applikationsserver auch als Container bezeichnet. Der Tomcat ist ein bekannter Vertreter unter den Applikationsservern und die Referenzimplementierung von SUN. Für diesen Server existieren zwei kombinierbare Wege, die Sicherheit des Servers zu erhöhen und ihn gegen Angriffe zu härten.

security Option In der Grundeinstellung des Tomcat unterliegt ein Programm keinerlei Begrenzung in der Verwendung von Resourcen. Durch die security Option wird der SecurityManager aktiviert, der es dem Entwickler ermöglicht, eine feinkörnige Beschränkung der Rechte des Programms zu entwerfen.
Die Rechte werden in der Datei catalina.policy verwaltet. Das Format entspricht der herkömmlichen Policydatei, wie sie in Java Verwendung findet. Hier können Rechte für eine gesamte Codebasis oder für einzelne Javaklassen gesetzt werden. Eine genaue Darstellung der Möglichkeiten findet sich in [JF03, Seite 133] oder [Fou].

chroot-Umgebung Unix und die Derivate davon bieten die Möglichkeit, über den chroot-Befehl ein Programm in einem anderen Verzeichnis oder unter einem anderen Benutzer auszuführen. Bei einem erfolgreichen Angriff auf den Dienst ist es dem Angreifer nicht möglich, auf wichtige Dateien des Betriebssystems Zugriff zu nehmen. Hierdurch kann der Angriff an sich nicht verhindert werden, aber eine Kompromittierung des Systems wird verhindert.

In [JF03] stellen J. Brittain und I. F. Darwin fest, dass eine Kombination aus dem Security Manager von Tomcat und chroot ein sehr hohes Sicherheitsniveau darstellt. Auch wenn der Security Manager sehr sicher ist, so kann es doch unentdeckte Sicherheitslücken geben, die einem Angreifer Zugriff auf das darunter liegende System gewähren könnte. Durch den Einsatz der chroot-Umgebung wird diesem vorgebeugt. Allerdings ist auch dies nicht absolut sicher, da auch das Betriebssystem Fehler enthalten kann.

Auch wenn sich die Entwickler größte Mühe gegeben haben, den Tomcat so sicher wie möglich zu machen, kann diese Sicherheit durch ein schlecht entwickeltes Programm beeinträchtigt werden. In [JF03] werden verschiedene Schwachstellen vorgestellt. Sie haben gemein, dass sie auf konstruierten Anfragen beruhen, mit dem Ziel, die Sicherheit des Tomcat zu umgehen.

Cross-site scripting Eine der häufigsten Attacken auf Webdienste ist das Cross-Site Scripting (XSS). Hierbei geht es darum, einen fremden Browser dazu

zu bringen, Skripte von einem anderen Server auszuführen und so Daten über den fremden Benutzer sammeln zu können.

Um einen Angriff durchführen zu können, sucht ein Angreifer einen Server auf dem er z.B. in Diskussionsforen folgendes hinterlegen kann:

```
< a href=''http://www.example.com/search?query=<script language='javascript'>document.
    location=''http://www.grovywings.com/foo'' + document.cookie<script>''>link</a>
```

Listing 15: Beispiel eines XSS Angriffs

Wenn ein Benutzer auf diesen Link klickt, wird der richtige Server mit dieser Suche beauftragt. Wenn nun der Server eine Antwortseite produziert, auf der auch der von Benutzer eingegebene Inhalt steht, wird der im Inhalt eingebettete Skriptbefehl ausgeführt. Der Benutzer sendet sich also selber bösartigen Code. Eine genauere Darstellung von XSS Angriffen kann in [Eck03, Seite 100], in [JF03, Seite 147] oder [Woe04] gefunden werden.

HTML Injection Hierbei wird HTML Code auf eine Seite gestellt, so dass Besucher etwas anderes sehen als der Autor der Seite wollte. In [JF03] wird festgestellt, dass die für HTML Injection anfälligen Seiten größtenteils ein HTTP-Get erlauben um HTML-Code hochzuladen, in der Länge, die der HTTP-Client in einer URL erlaubt. Leider wird dieser Angriff in der Literatur meist in Kombination mit XSS-Angriffen aufgeführt und nicht gesondert behandelt. Eine Darstellung des Angriffs findet sich in [JF03].

SQL Injection Hier geht es um einen Angriff auf die Datenbank, die hinter einem Webserver steht. Die Technik ist ähnlich dem XSS-Angriff. Meist findet diese Form des Angriffs über die Formfelder einer Webanwendung statt. Hierzu ist es nötig, dass die Daten, die dem HTML-Form übergeben werden, ungefiltert an die Applikation weitergereicht werden. Die Eingabe von „ ' or 0=0 - -" kann somit zu folgender resultierender SQL-Abfrage führen:

```
Select count(*) from users where username='' or 0=0 -- and password= limit 1
```

Listing 16: Beispiel eines SQL-Injection Angriffs

Da das „- -" eine SQL-Abfrage an dieser Stelle beendet und der String somit nicht mehr vollständig ausgewertet wird, hat sich der Angreifer erfolgreich an dem Server angemeldet. In [Woe04] wird das Thema ausführlich diskutiert.

Command Injection Bei diesem Angriff wird der Server dazu gebracht, an der Shell Befehle auszuführen. Dieses Problem kann in jedem Programm gefunden werden, dass auf Kommandozeilenprogramme zurückgreift.

Im Tomcatumfeld tritt dieses Problem nicht auf, solange nicht CGIServlet verwendet wird oder die Applikation selber entsprechende Tools verwendet.

Als ein gutes Mittel gegen diese Angriffe wird in [JF03] das HTTP Request Filtering angeführt. Hierbei wird die Interaktion zwischen Server und Client kontrolliert unter Verwendung der Tomcat Valves (vgl. [Fou04a]). Es ist natürlich auch möglich, eine entsprechende Kontrolle in AspectJ zu implementieren. Ein Ansatz der hier Verwendung finden kann, ist ein modifizierter HTMLRewriter, der im Kapitel 8.7 vorgestellt wird. Die Idee ist es, einen Filter vor die Methoden doGet, doPost und doPut zu setzen, so dass die ankommenden Strings vor einer Auswertung in der Software geprüft werden.

Ein Angriff der nicht durch Filter erkannt werden kann ist das Session Hijacking. Hier wird eine Sitzung durch einen Angreifer übernommen. Dies kann auf zwei verschiedene Weisen erfolgen, wie von J. Prosise in [Pro85b] angeführt wird: erraten einer ID oder das Session ID Cookie stehlen. Der erste Weg ist nicht erfolgversprechend, da die ID eine sehr grosse Zufallszahl ist. Der Angreifer muss somit in den Besitz der Cookies kommen. Um die Übertragung des Cookies zu schützen, muss diese krytografisch geschützt werden. Hier bietet sich SSL an. Aber auch dann kann das Cookie noch gestohlen werden. Hierzu führt J. Prosise XSS, Man-In-The-Middle Angriffe und physikalischen Zugriff auf den Client als Möglichkeiten an. Weiter stellt er fest, dass es gegenwärtig nicht möglich ist, einen solchen Angriff abzuwehren. Es kann lediglich dem Angreifer sehr schwer gemacht werden.

Ein weiterer wichtiger Baustein in der Absicherung des Tomcat ist die Verwendung von SSL zum Schutz der Kommunikation. Da die Kommunikation im Internet nicht vertraulich abläuft, ist jede Information frei lesbar. Um dies zu verhindern, muss ein sicherer Kanal geschaffen werden, in dem die Pakete verschlüsselt übertragen werden. Mehr dazu findet sich im Kapitel 8.8.1.

6.4 Beurteilung von Sicherheit

Um die Sicherheit eines Systems zu beurteilen und eine Aussage über das Sicherheitsniveau treffen zu können, wird eine methodische Vorgehensweise benötigt. Hierdurch können verschiedene Systeme untereinander verglichen werden. Wichtig ist es, dass die Methode erprobt ist und allgemein anerkannt, da auch hier Fehler in der Methode nicht ausgeschlossen werden können.

Aufgrund der verschiedenen Anforderungen haben sich über die Zeit auch verschiedene Kriterienkataloge entwickelt. Die wichtigsten Kataloge sind laut M. Moschgart in [Mos03]:

- D-BSI IT–Sicherheitshandbuch [fSidI92]
- D-BSI Grundschutzhandbuch [fSidI03b]
- GB-BSI BS 7799 „Code of Practice for information security management” [fSidI04]

- ISO/IEC 17799 „Information technology – Code of practice for information security management”[Gro04]
- ISO/IEC TR 13335-1 bis 5 „Information Technology – guidelines for the management of IT Security (GMITS)”
- RFC 2196 „Site Security Handbook” [Fra97]
- Cobit [AA04]

In [Eck03] werden noch folgende Kriterienkataloge angeführt:

- ITSEC (Information Technology Security Evaluation Citeria) [fSidI97]
- TCSEC (Trusted Computer Systems Evaluation Criteria) [STA85]

Das BS 7799 wird vom englischen BSI herausgegeben und wurde 1999 in einer zweiten überarbeiteten Version veröffentlicht, die international anerkannt ist. Aus diesem Standard ist im Dezember 2000 der ISO/IEC 17799 hervorgegangen. Die ISO hat diesen Standard weiterentwickelt. Er stellt eine Sammlung von Empfehlungen für die Informationssicherheit dar, also Verfahren und Methoden, die sich in der Praxis bewährt haben. Es werden aber keine konkreten Maßnahmen empfohlen, so dass er flexibel bleibt.

Bei der ISO/IEC TR 13335 werden hingegen keine Vorgehensweisen oder Lösungen vorgegeben. Er ist ein Leitfaden, wie Vorgehensweisen zu entwickeln und anzupassen sind. Daher ist er auch nicht in der Lage, ein Sicherheitsniveau zu messen oder in Relation zu anderen Systemen zu stellen.

Im RFC 2196 werden Sicherheitsrichtlinien im Internet beschrieben und richtet sich speziell an Systembetreiber.

In Deutschland ist 1992 vom BSI das Sicherheitshandbuch in seiner ersten Auflage herausgegeben worden. Dieses stellt einen Leitfaden dar, anhand dessen eine Risikoanalyse durchführbar ist. Doch der Aufwand hierfür ist sehr groß, so dass im Juli 1994 die erste Version des Grundschutzhandbuchs (GSHB) herausgegeben wurde. Dieses wird seitdem jährlich aktualisiert. Es ersetzt die Risikoanalyse des Sicherheitshandbuchs durch einen Ist-Soll-Vergleich.

In einer Sicherheitsanalyse werden alle Objekte, die bedroht werden, erfasst und für jedes Objekt werden die Grundbedrohungen sowie die Gefährdungen ermittelt. Daraufhin wird jedes Objekt bewertet und die Häufigkeit der Schäden ermittelt. Aus diesem Resultat werden dann Maßnahmen abgeleitet, wie ein einzelnes Objekt zu schützen ist. Dieser Prozess ist sehr aufwendig und für viele Firmen nicht durchführbar. Daher setzt das Grundschutzhandbuch auf einen Soll-Ist-Vergleich. Damit wird versucht durch Schablonen ein System zu erfassen und konkrete Handlungsanweisungen aus einem Katalog herauszusuchen. Nach dem Ausführen der Maßnahmen kann davon ausgegangen werden, dass das System

genau so sicher ist wie das Referenzsystem, welches dem Grundschutzhandbuch zugrunde liegt.

Falls festgestellt wurde, dass ein höherer Schutzbedarf vorhanden ist, als ihn das Grundschutzhandbuch bieten kann, so sind ergänzende Sicherheitsanalysen vorgesehen. Diese werden auf Grund der hohen Kosten auf die sicherheitskritischen Bereiche beschränkt. Folgende Methoden sind vorgesehen:

Risikoanalyse Eine Risikoanalyse, wie sie schon weiter oben vorgestellt wurde, erfasst für jedes betrachtete Objekt die Bedrohungen und Gefährdungen. Die durch das GSHB beschriebene Risikoanalyse beschränkt sich auf die Bereiche des Systems, für die das GSHB nicht ausreichend ist. Somit ist der Aufwand bedeutend geringer.

Penetrationstest Es wird ein Angriff auf das System simuliert. Hierbei wird zwischen einem Blackbox- und einem Whiteboxangriff unterschieden. Bei einem Blackboxangriff besitzt der Angreifer keinerlei Wissen über das vorhandene System. Der Whiteboxangriff hingegen setzt bei dem Angreifer ein detailliertes Wissen über interne Bedingungen voraus. Hierdurch kann ermittelt werden, welche Schwachstellen vorhanden sind und welche Schäden ein Angriff erzeugen kann.

Differenz-Sicherheitsanalyse Dieses Verfahren vergleicht das zu schützende Teilsystem mit Musterlösungen, die sich für hochschutzbedürftige Systeme etabliert haben.

Im GSHB [fSidI03b] wird ausgeführt:

> Typische höherwertige Maßnahmen im Bereich der IT-Systeme sind der Einsatz von zertifizierten Betriebssystemen oder von speziellen Sicherheitsversionen der Betriebssysteme, der Einsatz von Authentisierungstoken oder sogar die Isolation der IT-Systeme. Im Bereich der Kommunikationsverbindungen kommen beispielsweise folgende höherwertige Maßnahmen zum Einsatz: Kappung von Außenverbindungen, Leitungs- oder Ende-zu-Ende-Verschlüsselung, gepanzerte Kabeltrassen oder drucküberwachte Kabel, redundante Kommunikationsstrecken oder redundante Kabelführung sowie Einsatz von mehrstufigen Firewalls kombiniert mit Intrusion Detection Tools. Im Bereich der infrastrukturellen Sicherheit können beispielsweise Vereinzelungsschleusen, Brandlöschtechnik, Videoüberwachung, Zutrittskontrollsysteme und Einbruchmeldeanlagen bis hin zu Backup-Rechenzentren eingesetzt werden.

Hierzu existieren Schutzklassenmodelle des BSI für thematisch eingegrenzte Bereiche.

Durch die Einführung der ITSEC Kriterien wurde ein harmonisiertes Regelwerk zwischen Großbritannien, Frankreich, den Niederlanden und Deutschland geschaffen. Es gibt sieben Evaluationsstufen (E0 bis E6) durch die der Grad an Vertrauen ausgedrückt wird, den ein System erreicht hat. E0 stellt ein System mit unzureichender Vertrauenswürdigkeit dar. Darauf aufbauend definieren sich die höheren Stufen:

E1 Die Sicherheitsanforderungen müssen formuliert sein. Es muss eine Sicherheitsstrategie existieren und die Bedrohungen für das System sind erfasst.

E2 Es wird eine Testdokumentation gefordert und eine Bewertung der Tests im Hinblick auf die geforderten Sicherheitsanforderungen.

E3 Es werden strengere Anforderungen an die Implementierung gestellt. Es muss der Zusammenhang zwischen der Implementierung und dem Feinentwurf hergestellt werden.

E4 Ein formales Sicherheitsmodell der Sicherheitseigenschaften wird gefordert. Ausserdem muss der Architekturentwurf und der Feinentwurf in einer semiformalen Notation beschrieben sein.

E5 Ab dieser Stufe wird eine erklärende Dokumentation des Systems gefordert. Es muss der Zusammenhang zwischen Quellcode und dem Feinentwurf erklärend dargestellt werden.

E6 Für die höchste Evaluationsstufe werden mathematisch fundierte Beweisverfahren gefordert.

Neben der Stufe ist noch die Qualität entscheidend. Diese wird in niedrig, mittel und hoch eingestuft. Somit wird ein System durch ein Paar, welches aus Evaluationsstufe und Qualität besteht, beurteilt.

7 Konzept und Sicherheitsbetrachtung

Die in den vorherigen Kapiteln vorgestellten Techniken sollen innerhalb eines Gesamtkonzeptes miteinander verbunden werden, so dass eine beliebige in Java programmierte Webanwendung nachträglich abgesichert werden kann. Um zu zeigen, das dieses Konzept tragfähig ist, wird es in Form eines Demonstrators implementiert (siehe Kapitel 8).

7.1 Konzept

Es soll eine generische Absicherung eines Webservices realisiert werden. Das eigentliche Produkt muss in diesem Fall nicht im Sourcecode vorliegen, sondern wird lediglich um eine neue Funktionalität erweitert. Diese Funktionalität besteht im Hinzufügen von Autorisations- und Authentifikationsmechanismen. Ein besonderes Augenmerk liegt in der räumlichen Verteilbarkeit dieser Dienste auf verschiedene Server. Ein weiterer Bereich ist das Überwachen und Führen der Benutzer in der Verwendung von Hardwaretoken, die zur Authentifikation Verwendung dienen.

Der zentrale Teil ist das Sicherheitsmodul, in dem die Autorisation und Authentifikation implementiert ist und welches zu einer Wirtsapplikation hinzugefügt werden kann.

Da der Sourcecode der Wirtsapplikation nicht vorliegt, muss das System auf andere Weise erfahren, welcher Benutzer welche Resourcen verwenden darf. Hierzu wird eine Trainingsphase der eigentlichen Produktivitätsphase vorgeschaltet, in der das Sicherheitsmodul an das zu schützende System angepasst wird.

Ein weiteres Ziel besteht in der Definition eines XML-basierenden Datenformats, mit dessen Hilfe die Sicherheitseigenschaften beschrieben werden können, und der Entwicklung eines Tools zur halbautomatischen Erzeugung der Beschreibung.

Die Vorteile dieses Konzepts liegen in seiner Modularität. Das Sicherheitsmodul kann als eigenständiges Projekt entwickelt werden und in verschiedenen Applikationen zum Einsatz kommen. Hierdurch senken sich die Entwicklungskosten für diesen Bereich der Software. Gleichzeitig sind weniger Fehler in dem Sicherheitsmodul zu erwarten, da es eine abgeschlossene Einheit darstellt, welches intensiv getestet werden kann. Auch wird durch die Entfernung von Sicherheitssourcecode aus der Wirtsapplikation diese einfacher Strukturiert. Auch dies reduziert die Kosten in der Entwicklung der Software. Falls durch Veränderungen der Organisationsstrukturen (im Einsatz der Wirtsapplikation) Veränderungen am Sicherheitsmodul erforderlich machen, so kann durch einen Austausch des Sicherheitsmoduls auf einfache Art und Weise eine neue Sicherheitsarchitektur wie z.B. Chinese Wall eingeführt werden. Durch die Trainierbarkeit wird ebenfalls möglichen Änderungen der Organisationsstruktur Rechnung getragen, da die vorgegebenen Arbeitsabläufe (Workflows) jederzeit angepasst, gelöscht oder

neu erstellt werden können und dadurch flexibel an die Bedingungen angepasst werden.

Authentifikation Für die Authentifikation findet das in Kapitel 2.2 beschriebene Liberty-Protokoll Anwendung, dessen Einsatz in Kapitel 2.1.6 motiviert wird.

Durch die Trennung von Applikation und Loginserver ist es möglich, den Loginserver als eigenes Schutzobjekt zu betrachten. Dadurch wird es möglich für die Verwaltung der Authentifikationsmerkmale getrennte Server und Administratoren einzusetzen. Auch kann dieser Teil vollkommen getrennt von der Applikation weiterentwickelt werden. Hierbei ist zu beachten, dass nach einer erfolgreichen initialen Anmeldung an den Server die Software eine regelmässige Reauthentifikation am Server vorsehen kann. Auf diesem Weg kann die Gültigkeit einer ausgestellten Authentifikation regelmäßig durch den Identity Provider neu geprüft werden.

Autorisation Nach einer erfolgreichen Authentifikation am Loginserver stehen dem Benutzer potentiell alle Resourcen der Software zur Verfügung. Eine mögliche Einschränkung ist, die Interaktion des Benutzers mit der Software zu reglementieren. Hierzu führen wir den Begriff der Workflows ein wie sie in Kapitel 4.3 definiert werden. Jeder Workflow beschreibt eine Abfolge von Webseiten und den Benutzereingaben, die erlaubt sind, um von einer Seite zur Nächsten zu kommen. Dies lässt sich über eine Finite State Machine beschreiben. Die Übergänge können durch reguläre Ausdrücke erfasst werden.

Als Sicherheitsmodell findet RBAC Anwendung. Dieses wird in Kapitel 4.2 eingeführt. Die Konsequenz hieraus ist, dass jede Interaktion, die nicht ausdrücklich erlaubt ist, als verboten eingestuft wird. Die Entscheidungsbasis liefert eine Datenbank, in der die Workflows sowie die Rollen und Benutzer als relationales Schema gespeichert sind. Ein ähnliches Konzept wird in [ASKP00] verfolgt.

Proaktivität Die Grundbausteine eines proaktivem Systems sind nach A. Salovaara und A. Oulasvirta [SO04]:

> 1) working on behalf of, or pro, the user, and 2) acting on their own initiative

Ein proaktives System ist also ein System, das für einen Benutzer arbeitet aber auch aus Eigeninitiative heraus handeln kann. Die Autoren sehen Proaktivität dabei als ein Teil von „context-awareness", also der Fähigkeit eines Programms sich seiner Umgebung „bewusst" zu sein.

David Tennenhouse beschreibt ein proaktives System in einer mehr technischen Sicht in [Ten00] als:

> Proactive systems will be intimately connected to the world around them, using sensors and actuators to both monitor and shape their physical surroundings.

Ein proaktives System verfügt hiernach über Sensoren und tätigt selbstständig im Sinne des Benutzers Überwachungen, trifft Entscheidungen und führt Aktionen aus. Der Benutzer steht dabei sowohl über dem proaktiven System im Sinne eines Überwachers als auch innerhalb des Wirkungskreises des Systems, in dem sowohl der Benutzer direkt auf das proaktive System einwirken kann als auch das proaktive System auf den Benutzer.

Die Verwendung von Hardwaretoken als Merkmal zur Authentifikation ist mit dem Problem behaftet, dass jede Person, die den Token besitzt auch vom System als legitimer Benutzer akzeptiert wird. Deshalb sollte eine initiale Authentifikation nur aus einer Kombination mit einem anderen Authentifikationsverfahren durchgeführt werden, z.B. zusätzlich mit einer Passworteingabe.Damit wird eine Authentifikation nach Wissen und Besitz durchgeführt (siehe Kapitel 5). Außerdem darf ein solcher Token nicht Dritten leichtfertig verfügbar gemacht werden. Da dieses Problem bei vielen Benutzern nicht als bekannt vorausgesetzt werden kann, ist es erforderlich, die Benutzer in diesem Themenbereich zu sensibilisieren.

Falls ein Token, der der Identifikation dient, an einem unbeobachteten System zurückgelassen wird, besteht die Gefahr, dass sich jemand Drittes Zugang zu dem zu schützenden System verschafft. Dies kann durch einen zuvor erfolgen Passwortdiebstahl oder eine noch angemeldete Session erreicht werden. Der Angreifer kann dann im Namen eines Mitarbeiters in dem System arbeiten. Hier greift die Proaktivität und soll den Benutzer daran erinnern, den Token nicht zurückzulassen und damit Angriffe dieser Art verhindern helfen.

Training Da dem Sicherheitsmodul die Abläufe in der zu schützenden Applikation nicht bekannt sind, müssen diese erst erlernt werden. Dies geschieht durch das Durchführen der Workflows unter speziell kontrollierten Bedingungen. Diese müssen danach durch einen Administrator von dem speziellen Fall auf die allgemeinen Workflows generalisiert werden. Damit ist gemeint, dass wenn z.B. beim Durchführen einer Überweisung ein Limit von 2000 Euro besteht, so muss dies als Regel definiert werden, die die spezielle Eingabe (2000 Euro) ersetzt. Die spezielle Eingabe ist durch die exemplarische Benutzung erzeugt worden. Als Mittel zur Formulierung dieser Regeln kommen reguläre Ausdrücke zur Anwendung.

Weitere Möglichkeiten ergeben sich durch eine Interaktion von Sicherheitsmodul und Wirtsapplikation, da hierdurch das Sicherheitsmodul auf interne Zustände der Wirtsapplikation reagieren und in Abhängigkeit von diesen Zuständen den Resourcenzugang beschränken kann.

Gesamtkonzept Das Zusammenwirken der vorgestellten Komponenten wird in Abbildung 21 illustriert. Der Identity Provider und das Sicherheitsmodul be-

sitzen jeweils eine Datenbank, die aus der durch die Trainingssuite erzeugten XML-Daten erzeugt werden. Die Aufgaben der Datenbanken teilen sich in die oben beschriebenen Bereiche Authentifikation und Autorisation. Die Datenbank des IDPs trägt alle Informationen, die nötig sind, einen Benutzer zu authentifizieren. In der Datenbank des Sicherheitsmoduls werden alle Daten gelagert, die benötigt werden, um eine Authentifikation auszulösen und danach die Handlungen eines Benutzers kontrollieren zu können. Auf dem Client wird der Hardwaretoken

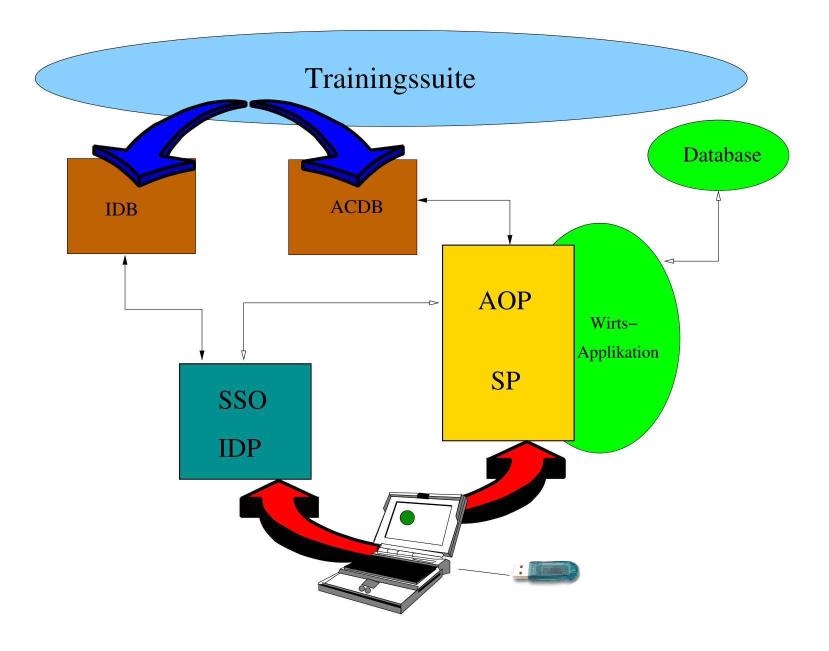


Abbildung 21: Gesamtkonzept

durch ein Applet angesprochen, das den Kontakt mit dem IDP aufbaut. Daneben wird noch ein zweites Applet gestartet, welches die Proaktivität des Systems zur Verfügung stellt und den Benutzer informiert.

7.2 Feinkonzepte

Im Folgenden werden einige Details des vorgestellten Konzepts näher betrachtet. Das Sicherheitsmodul wird durch den SecurityAspect in AspectJ (vgl. Kapitel 3) implementiert.

Veränderung der Wirtsapplikation Jede Kommunikation der Wirtsapplikation wird durch die in dieser Diplomarbeit entwickelten Aspecte verarbeitet. Die benutzerseitigen Eingaben werden, wie in Abbildung 22 zu sehen, durch den SecurityAspect behandelt. Die durch die Wirtsapplikation erzeugten Antwortseiten werden durch den HTMLRewriter bearbeitet.

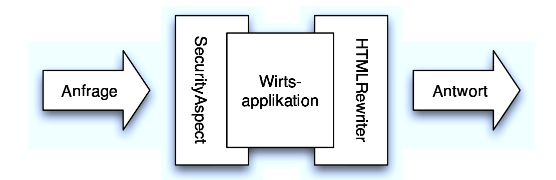


Abbildung 22: Aspecte verbunden mit der Wirtsapplikation

Zusammenspiel des Sicherheitsmoduls mit der Wirtsapplikation In Abbildung 23 wird die Kommunikation skizziert, die bei der ersten Anmeldung am zu schützenden Programm stattfindet. Es ist zu sehen, dass jeder Kontakt durch den securityAspect entgegengenommen wird. Dieser entscheidet, ob eine Authentifikation oder Reauthentifikation benötigt wird und ob der Zugriff auf das zu schützende System gewährt wird. Die genauen Abläufe, die für die Durch-

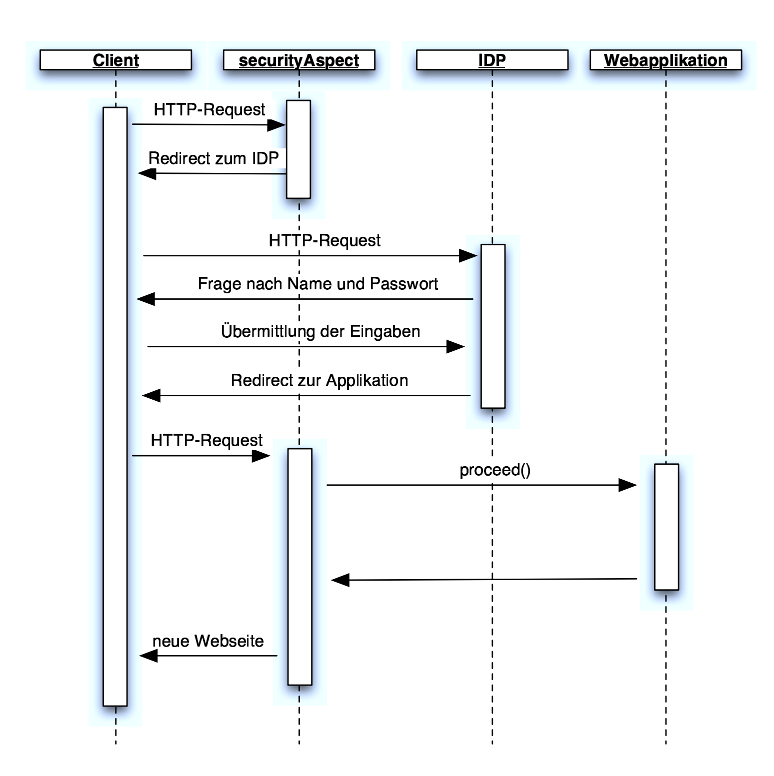


Abbildung 23: Ablauf einer Anmeldung

führung des Single-Sign-On zum Tragen kommen, werden im Kapitel 2.2 ab Seite 18 beschrieben.

Der in Abbildung 23 skizzierte Abblauf zeigt, dass das Sicherheitsmodul bzw. SecurityAspect die zentrale Rolle in diesem Konzept einnimmt. Alle Maßnahmen wie Authentifikation, Reauthentifikation und Autorisation werden durch den SecurityAspect entweder ausgelöst oder durchgeführt. Um eine bessere Verwend-

barkeit des Konzepts zu erreichen, erzeugt er ebenfalls Fehlermeldungen und kann, bei Bedarf, auch eine Anmeldung des Benutzers an der zu schützenden Applikation durchführen. Besonders der letzt genannte Punkt ist für eine hohe Unabhängigkeit des SecurityAspect von der Applikation notwendig.

Die Authentifikation wird nicht durch den SecurityAspect direkt durchgeführt, sondern durch das Liberty Single-Sign-On Framework. Der SecurityAspect führt eine HTML-Weiterleitung zu einem Identity Provider (IDP) durch. Dieser fragt nach einem Benutzernamen und einem dazu passenden Passwort. Danach wird wieder zu dem SecurityAspect weitergeleitet. Falls die Anmeldung erfolgreich ist wird geprüft in Abhängigkeit der Rollenmitgliedschaft, ob ein Hardwaretoken abgefragt werden muss. Falls eine Rolle, der der Benutzer zugehörig ist, eine entsprechende Überprüfung fordert, wird wieder an den IDP weitergeleitet, so dass der IDP auf die Existenz des Tokens testet. Dies setzt eine Möglichkeit zum Auslesen der Hardware am Client voraus. Im Gegensatz zu einer Passwortabfrage, die durch den Browser durchgeführt werden kann, ist dafür eine spezielle Software notwendig. Die Verbindung zwischen dem Hardwareinterface und dem IDP wird durch ein signiertes Browserapplet durchgeführt. Es leitet die Anfragen des IDP zu der Schnittstelle des Hardwaretoken weiter.

Modularität der Proaktivität Da ein Hardwaretoken leicht durch den Benutzer vergessen werden kann, muss die Software den Benutzer an diesen Token erinnern. Dies erfolgt ebenfalls durch ein Applet. Damit dieses Applet durch den Browser geladen und ausgeführt wird, muss das Applet in die Webseite der zu schützenden Applikation eingebaut werden. Die Anwendungsunabhängigkeit des Konzepts wird durch den HTMLRewriterAspect gewahrt. Dieser modifiziert die erzeugten Webseiten und fügt das Applet hinzu.

Kommunikation von IDP und Sicherheitsmodul Diese Kommunikation zwischen Client und dem IDP bzw. dem SecurityAspect muss durch entsprechende kryptografische Maßnahmen gegen Angriffe gesichert werden. Dies wird im Kapitel 7.3 und 2.5 näher betrachtet.

Daten in den Sicherheitsdatenbanken Durch den Einsatz eines Single-Sign-On Protokolls ergibt sich eine, auch räumliche, Trennung des Dienstes von der Anmeldung der Benutzer. Der Dienst, wie auch der Identity Provider, benötigen daher getrennte Datenbanken, in denen lediglich die Informationen hinterlegt sind, die für ihre Aufgabe benötigt werden.

Der Identity Provider benötigt hierbei die Benutzernamen und Authentifikationsmerkmale der Benutzer. Das Sicherheitsmodul, welches den eigentlichen Dienst umschliesst, benötigt die Informationen über die Workflows und die Authentifikationsmerkmale, die für eine Rolle bzw. Benutzer verlangt werden.

Darüber hinaus muss es noch Tabellen geben, die den aktuellen Zustand für eine Session festhalten.

Interaktion des Sicherheitsmoduls mit der Wirtsapplikation Die Interaktion wird durch ein Ansprechen der Wirtsdatenbank erreicht, so dass in der XML-Beschreibung (8.2) SQL-Anfragen formuliert werden können, deren Ergebnisse als Entscheidungsgrundlage herangezogen werden. Hierbei wird davon ausgegangen, dass eine Wirtsapplikation alle Daten in einer entsprechenden Datenbank verwahrt. Eine Erweiterung dieses Konzepts auf das Auswerten von z.B. Konfigurationsdateien ist möglich.

Somit ergibt sich ein Gesamtbild, welches in Abbildung 24 dargestellt ist. In ihm werden von links nach rechts der Client, der IDP und der SP dargestellt. Der SP besteht aus dem SecurityAspect und der Wirtsapplikation. Die einzelnen Module haben jeweils in der Vertikalen miteinander Kommunikationsbeziehungen. Die Farben sind in Anlehnung an Abbildung 21 gewählt. Grün eingefärbt sind die Komponenten der Wirtsapplikation, gelb die Service Provider Module, türkis alle Komponenten des IDPs sowie der Proaktivität und braun die Datenbanken des IDPs und SPs.

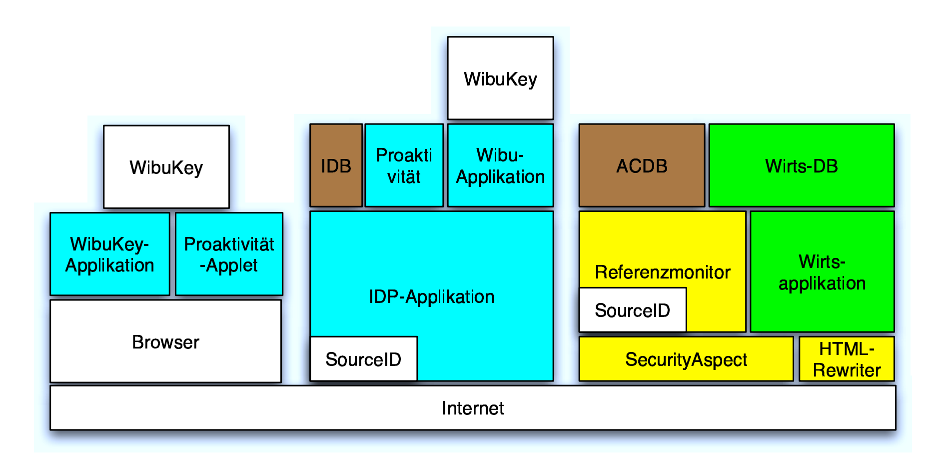


Abbildung 24: Gesamtkonzept

7.3 Sicherheitsbetrachtung des Konzepts

Eine Sicherheitsbetrachtung eines aus Sicherheitsmodul und Wirtsapplikation zusammengesetzten Systems ist im Allgemeinen nur schwer möglich, da die Eigenschaften des Gesamtsystems nicht nur von dem hier vorgestellten Konzept abhängig sind, sondern auch von dem zu schützenden System. Sicherheitsverletzungen die durch die Wirtsapplikation entstehen, können durch das hier vorgestellte Konzept nicht erkannt werden. Die Grenzen das Ansatzes stellen sich wie folgt dar:

- Das Sicherheitsmodul hat Konzeptbedingt keine Möglichkeit zu überprüfen, ob die Antworten der Software den gewollten Antworten entsprechen. Die in Kapitel 6.3 vorgestellten Angriffe werden durch den SecurityAspect nicht verhindert. Eine inhaltsbasierte Kontrolle von Informationsflüssen kann nur bedingt erfolgen, da das Sicherheitsmodul nur auf der Basis der Eingaben des Benutzers entscheidet, ob ein nachfolgender Zustand legitim ist.

- Es wird nicht verifiziert, dass das zugrunde liegende System unverändert ist. Eine Veränderung der Klassen des Systems wird nicht erkannt. Dies wird in Kapitel 10 nochmals beleuchtet.

- Das Konzept ist auf die Absicherung der Server angewiesen. Sowohl die Identity Provider Applikation, die die Authentifikationen durchführt, als auch die Service Provider Applikation, der SecurityAspect, sind nur eine Tomcat Applikation. Tomcat läuft auf einem Server und dieser Server muß abgesichert sein (siehe Kapitel 6).

- Das Konzept hat die Annahme, dass der Austausch der Authentifikationsdaten durch das Liberty Protokoll abgesichert ist. Eine Sicherheitsbetrachtung des Liberty Protokoll befindet sich in Kapitel 2.5.

- Die Sicherheit der Wibukey-Authentifikation basiert auf der Zusicherung durch die Firma Wibu-Systems, dass sichergestellt ist, dass ein Firm-Code eindeutig und jedem Kunden speziell zugeordnet ist. Eine genauere Betrachtung des Wibu-Key findet sich in Kapitel 5.3.

- Das Konzept des ProAktivApplets basiert auf dem Zusammenspiel der Service Provider und Identity Provider. Der Aufruf des Applets muß vom Service Provider in jede Ausgabeseite eingebettet und das Applet direkt vom Identity Provider geladen werden. Falls einer der beiden Provider nicht mit den nötigen Modulen für die Proaktivität ausgestattet ist, kann das ProAktivApplet nicht ausgeführt werden (vgl. Kap. 8.10).

Aus dem Konzept ergeben sich auch die zu betrachtenden Angriffspunkte, die im Rahmen einer Sicherheitsanalyse, wie sie in Kapitel 6.4 beschrieben wird, betrachtet werden müssen. Für jedes Element des Konzepts muss daher untersucht werden, wie und mit welchen Auswirkungen ein Angriff erfolgen kann:

Service Provider-Datenbank In der Service Provider Datenbank sind, wie in Kapitel 7.2 beschrieben, alle Informationen hinterlegt, die für die Autorisation benötigt werden. Zusätzlich befinden sich hier auch die für eine Rolle zu verlangenden Authentifikationsverfahren. Die Kommunikation zwischen SecurityAspect und der Datenbank erfolgt über TCP/IP. Ein erfolgreicher Angriff auf diese Datenbank kann (1) ein Offenlegen der Datenbank zur Folge haben oder (2) eine Veränderung der gespeicherten Daten. Im Fall

(1) werden die Workflows und damit auch die Rechte der einzelnen Benutzer dem Angreifer offenbart. Darüber kann der Angreifer Einblicke in die Firmenorganisation bekommen. Da keine Passwörter oder andere Authentifikationsmerkmale in dieser Datenbank hinterlegt sind, bleibt der Schaden der durch einen lesenden Zugriff entsteht begrenzt. Der Fall (2) hat zur Konsequenz, dass die Rechte die ein Benutzer hat verändert werden und auch die benötigten Authentifikationsmerkmale durch den Angreifer geändert werden können. Er hat dadurch die Möglichkeit sich selbst Zugriff auf das System und dessen Daten zu gewähren und kann regulären Benutzern den Zugriff unmöglich machen.

Identity Provider-Datenbank Die Identity Provider Datenbank trägt die Daten zur Authentifikation der Benutzer. Im Gegensatz zur SP-Datenbank bedeutet der Fall (1), also der lesende Zugriff, bereits eine extreme Verletzung des Sicherheitsmodells.

Hieraus ergibt sich die Notwendigkeit die Datenbanken unzugänglich für externe Personen zu halten. Hierzu kann das in Abbildung 25 illustrierte

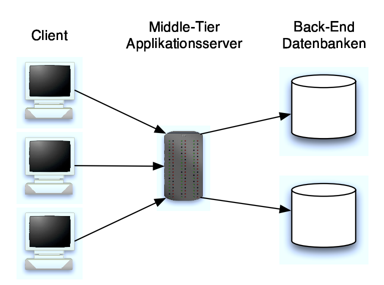


Abbildung 25: Drei Schichten Modell

Three-Tier Modell verwendet werden (vgl. [Wik]). Eine mögliche Massnahme sind Firewalls, die den Zugriff auf die entsprechenden IP-Ports verhindern.

Identity Provider Der Identity Provider ist das zentrale Modul, in dem die Authentifikation für verschiedene Dienste durchgeführt wird. Ein Ausfall des Identity Providers hat zur Folge, dass die auf ihm basierenden Dienste nicht weiter verwendbar sind, da für keinen Benutzer eine Authentifikation durchführbar ist. Er stellt eine ideales Ziel für eine Denial-Of-Service Attacke dar.

Aufgrund dieser zentralen Bedeutung gilt es, den Server der den Identity Provider beheimatet speziell abzusichern (siehe 6.1). Da der IDP in Java realisiert ist, müssen die in Kapitel 3.6 getroffenen Betrachtungen zur Sicherheit von Java und die in Kapitel 6.3 aufgezeigten Angriffsmöglichkeiten auf eine Websoftware beachtet werden.

Da die erzeugte Webseite statisch ist, sind die im Kapitel 6.3 ausgeführten XSS Angriffe weniger wahrscheinlich. Dafür muss aber speziell die Möglichkeit von SQL und Command Injection Angriffen im Einzelfall untersucht werden.

securityAspect Für den SecurityAspect gelten prinzipiell die Überlegungen, die für den Identity Provider zutreffen. Da der SecurityAspect allerdings nur eine sehr eingeschränkte Benutzerinteraktion besitzt, sind hier SQL und Command Injection Angriffe ausgeschlossen.

ProAktivApplet Das ProAktivApplet hat im Rahmen des Demonstrators lediglich eine Protokollierungsfunktion. Eine Veränderung dieses Applets könnte zu einer Falschaufzeichnung des Benutzerverhaltens führen. Falls Entscheidungen aufgrund der Aussagen dieses Applets beim IDP getroffen werden sollen, könnte durch eine Abänderung des Applets ein Zeitfenster für Angriffe geöffnet werden. Ab dem Zeitraum der Entferung des Hardware-Tokens bis zur nächsten routinemäßigen Kontrolle des Tokens durch das WibukeyApplet könnte ein Vorhandensein des Hardwaretoken vorgetäuscht werden. Die Authentizität des Applet selbst wird sichergestellt durch eine Signierung des Applets durch den IDP und der durch SSL geschützten Übertragung an den Client-Rechner.

WibukeyApplet Das WibukeyApplet hat eine Vermittlerrolle zwischen dem IDP und dem Hardware-Token. Dieses Applet wird genauso wie das ProAktivApplet per SSL übertragen und vom IDP signiert.

Erzeugung der Datenbanken Die Datenbanken werden aus einer XML-Beschreibungsdatei initialisiert. Der Mitarbeiter, der diese Datei erzeugt und die Daten in dieser Datei pflegt, legt die Sicherheitsbestimmungen die für die Software gelten fest. Diesem Mitarbeiter wird daher in besonderer Weise vertraut. Alle anderen Maßnahmen dienen letztlich lediglich dazu, dass diese Vertrauensbasis nicht kompromittiert werden kann. Bei der Erzeugung sollte daher ein Mehraugenprinzip zur Anwendung kommen. Auch ist es wünschenswert, das die erzeugte Beschreibung durch eine formale Definition der Anforderungen verifiziert werden kann.

Nach der Betrachtung der Module des Konzepts müssen die Verbindungen zwischen den Modulen betrachtet werden:

IDP(securityAspect)-DB Bei der Kommunikation mit den Datenbanken wird TCP/IP verwendet. Eine Authentifikation gegenüber der Datenbank erfolg mit Hilfe von Benutzernamen und Passwort. Diese werden in Konfigurationsdateien gespeichert oder sind fest einkodiert in der Applikation. Somit sind sie offengelegt, falls ein Angriff auf den Server erfolgreich durchgeführt wurde.

Da die Datenbanken über TCP/IP zu erreichen sind, müssen hier spezielle Schutzmassnahmen durchgeführt werden, die verhindern, dass die Informationen in ihr Dritten zugängig werden. Mögliche Maßnahmen werden durch das GSHB (siehe Kapitel 6.4) vorgeschlagen. Die Wirksamkeit ist anhand der Kriterienkataloge, wie sie in Kapitel 6.4 vorgestellt werden, zu überprüfen.

Client-Server Bei der Kommunikation zwischen dem Browser des Clients und den jeweiligen Servern können verschiedene Angriffspunkte bestehen. Kapitel 6.3 versucht hierbei die häufigsten aufzuzeigen. Neben diesen Angriffen muss noch das Problem eines Denial-Of-Service Angriffs betrachtet werden. Hierbei wird versucht die Schnittstelle des Servers so zu belasten, dass die regulären Dienste nicht ausgeführt werden können. Die Daten der Server werden in diesen Szenario nicht berührt, allerdings ist der Dienst nicht zu erreichen. Falls auf diesem Weg der IDP unbrauchbar gemacht wird, so können auch alle Dienste die diesen verwenden nicht benutzt werden.

IDP-securityAspect Eine Kommunikation zwischen dem Identity Provider mit dem SecurityAspect findet nicht direkt statt. Der Browser des Clients wird durch ein HTTP-Redirect dazu gebracht mit dem jeweils anderen System Kontakt aufzunehmen. Hieraus ergibt sich die Anfälligkeit des Konzepts gegenüber des in Kapitel 6.3 beschriebenen Session Hijacking Angriffs. Ein Benutzer wird gegenüber IDP wie auch SecurityAspect durch die hinterlegten Cookies identifiziert. Bekommt ein Angreifer die hinterlegten Cookies, so kann er sich als der entsprechende Benutzer ausgeben.

IDP-Wibukeyapplet Das Applet, welches der Kommunikation des IDP mit dem lokalen WIBUKEY-Server dient, besitzt zwei Schnittstellen. Die Kommunikation mit dem IDP wird durch SSL geschützt. Das Applet kann außerdem lediglich mit dem Aussteller des Applets, in diesem Falle nur dem IDP, kommunizieren. Dies ist durch das Java-Sicherheitskonzept sichergestellt. Durch die Signierung des Applets überprüft der Browser automatisch, ob das Applet unverändert ist.

IDP-proaktiv Die Kommunikation zwischen dem Identity Provider und dem ProAktivApplet ist per SSL abgesichert. Das Applet greift über TCP/IP auf eine lokale Applikation zu, die den Kontakt mit dem WibuKey herstellt. Es muss überprüft werden, inwieweit eine externe Anwendung zugriff auf den TCP-Port hat und welche Konsequenzen sich daraus ergeben (siehe 8.9.1).

Neben den Angriffspunkten auf das Konzept müssen auch die verwendeten Technologien auf Schwachstellen untersucht werden werden. In dem betrachteten Konzept wird als Schlüsseltechnologie die aspektorientierte Programmierung in Form

von AspectJ verwendet. Eine Sicherheitsanalyse von AspectJ muss den Weg wie ein Aspect in ein Programm eingefügt wird und das Ergebnis dieses Deployments betrachten.

Bei der Betrachtung des Ergebnisses des Deployments können die in Kapitel 3.6 getroffenen Aussagen zur Sicherheit von Java herangezogen werden. Ein Programm, welches durch Aspecte verändert wurde, ist wieder ein normales Javaprogramm. Dieses wird durch jede Java-Konforme Laufzeitumgebung ausgeführt und benötigt an dieser keine speziellen Änderungen. Somit bestehen alle Aussagen zur Sicherheit von Java weiter. Ein Programm wird in einer Sandbox isoliert ausgeführt und erhält lediglich die Rechte auf Resourcen, die explizit vergeben sind. Dies setzt voraus, dass der SecurityManager von Java aktiviert und korrekt eingestellt ist. Eine globale Freigabe aller Resourcen für alle Javaprogramme ist möglich und hebelt dieses Konzept aus. Die Prüfungen über die Korrektheit der Syntax werden wie in Kapitel 3.6 beschrieben durchgeführt. Es ist somit sichergestellt, dass das ausgeführte Programm syntaktisch korrekt ist und die Schnittstellendefinitionen eingehalten werden.

Einem Javaprogramm ist es nicht anzusehen, ob es durch einen Weaver (siehe Kapitel 3.1.2) verändert wurde. Es kann somit jedes beliebige Programm in seiner Funktionalität geändert werden. Um diesem Problem zu begegnen sieht Java die Signatur der Programme vor. Die Klassen werden hierzu in ZIP komprimierte Archive (JAR) zusammengefasst und danach signiert (siehe hierzu [SUN85] und [ST85]). Ein Benutzer kann dadurch kontrollieren, von wem die Software erzeugt wurde und ob sie seit der Signierung verändert wurde. Dieses Verfahren ist auch für die Web Archive (WAR) anwendbar, die im J2EE-Umfeld zum Einsatz kommen.

Der Inhalt einer war-Datei wird beim Deployment im Applikationsserver entpackt und ab diesem Moment nicht mehr verwendet. Der Server kontrolliert nicht, ob das Verzeichnis identisch zu dem Archiv ist. Ein Angreifer kann somit, falls er ungehinderten Zugriff auf den Server bekommt, über die aspektorientierte Programmierung das Programm in seinem Verhalten verändern oder, ähnlich dem beschriebenen HTMLRewriter, die Benutzerinteraktion protokollieren. Eine solche Veränderung kann beispielsweise durch ein Programm wie Tripwire (siehe Kapitel 6.2) erkannt werden.

Anhand dieser Sicherheitsbetrachtung wird es deutlich, dass es zur Zeit nicht möglich ist eine endgültige Aussage über die Sicherheit eines mit der hier vorgestellten Methode abgesicherten Webservices treffen zu können. Bekannte Angriffe auf Webservices können, wie in 6.3 gezeigt, abgewehrt werden. Aber zum Beispiel Session Hijacking Angriffe stellen ein großes Problem dar, da es keine Möglichkeit gibt, diese zu unterbinden. Es kann einem Angreifer lediglich sehr schwer gemacht werden, an die Cookies heranzukommen. Auch hängt jede Aussage über die Sicherheit des Konzepts von verschiedenen Voraussetzungen ab, deren Korrektheit nur im Einzelfall oder gar nicht überprüft werden kann. Diese sind die Integrität des Servers, die Korrektheit der Javasprachdefinition, die Korrektheit der Imple-

mentierung der Sprachdefinition und der Korrektheit des Applikationsservers.

Eine Sicherheitsaussage kann somit nur über pragmatische Vorgehensweisen getroffen werden, wie sie z.B. im Rahmen des GSHB beschrieben werden.

8 Spezifikation und Implementierung

Um das in Kapitel 7 beschriebene Konzept in Form eines Demonstrators umzusetzen, wurden fünf voneinander getrennte Module entwickelt, die die einzelnen Aufgaben übernehmen. Diese Module werden nun kurz vorgestellt und in den folgenden Abschnitten erklärt.

SecurityAspect Der SecurityAspect (ab Seite 98) umhüllt die Ausführung der eigentlichen Actionklassen. Eine Actionklasse stellt die Schnittstelle zwischen dem Applikationsserver und der Applikation dar, durch die die Benutzereingaben der Applikation übergeben werden. Innerhalb des Aspectes werden alle Aktionen, die zur Authentifikation und Autorisation notwendig sind, ausgelöst und, soweit es im SecurityAspect möglich ist, auch durchgeführt:

- Feststellen, ob sich der Benutzer bereits angemeldet hat. Falls dies nicht passiert ist, wird der Benutzer an den IDP weitergeleitet.
- Den Benutzer bei der Applikation anmelden.
- Feststellen, ob eine Reauthentifikation nötig ist.
- Kontrollieren, ob die vom Benutzer gewünschte Aktion zulässig ist.

HtmlRewriter Dieser Aspect, wie er auf Seite 104 beschrieben wird, dient der dynamischen Einbettung der Debuginformationen sowie des ProAktivApplets in eine erzeugte Webseite. Dazu verändert der Aspect die durch die JSP-Klassen der Wirtsapplikation erzeugten HTML-Seiten.

Identity Provider Der Identity Provider ist ein Single-Sign-On Server und stellt die Dienste im Rahmen des Frameworks für die Authentifikation eines Benutzers zur Verfügung. Dieses Framework wurde durch folgende Module aufgebaut bzw. erweitert, so dass spezielle Authentifikationsverfahren zur Verfügung stehen sowie ein proaktives Verfahren, welches den Benutzer an seinen Hardwaretoken erinnert.

- Integration eines **Identity Providers** in eine Struts Umgebung (Seite 105)
- Einbindung einer **Kanalabsicherung** (Seite 107)
- Einbindung des **Wibukeys** als Authentifikationsverfahren (Seite 110)
- Das **WibukeyApplet** stellt das clientseitige Interface zur Verifikation des Wibukeys zur Verfügung (Seite 112).
- Die Proaktivität, implementiert durch das **ProAktivApplet** (Seite 78).

Trainingssuite Das Erlernen eines Workflows teilt sich in zwei Module auf.

- Im **Webtool** (siehe 8.1) werden die Workflows angelegt, sowie die Benutzer und Rollen. Wenn ein Workflow angelegt wird und die Trainingsphase gestartet ist
- wird durch den **Trainingsaspect** die Interaktion des Trainers (Administrators) mit der zu schützenden Software aufgezeichnet und dem Webtool zur Verfügung gestellt.

Aus der Trainingssuite erhält der Trainer eine XML-Datei, in der alle Benutzer inklusive ihrer Authentifikationsmerkmale sowie Rollenzugehörigkeiten und Workflows enthalten sind. Diese kann manuell nachbearbeitet und später durch das Datenbanktool dem SecurityAspect verfügbar gemacht werden.

Datenbanktool Um die in der Trainingssuite erzeugten xml-Daten in die Datenbank des SecurityAspects zu transferieren, wird das Datenbanktool (dbtool) verwendet. Es erzeugt eine relationale Datenbankstruktur.



Abbildung 26: Entwicklungsschritte

Um eine Applikation zu schützen müssen nun folgende Schritte durchgeführt werden, die grob wie in Abb. 26 unterteilt werden können.

1. Der Trainungsaspekt muss unter Hilfe des AspectJ Weavers mit der Applikation verwoben werden. Danach werden mit dem
2. Webtool die Workflows angelegt und jeweils durch Benutzung der Applikation definiert. Daraufhin werden sie nachbearbeitet und die
3. Constraints über die Benutzereingaben definiert. Danach müssen die
4. User und Roles angelegt werden. Dabei müssen die Workflows den Rollen und Rollen den Usern zugeordnet werden. Bei den Benutzern müssen daneben noch die
5. Authentifikationsmerkmale definiert werden. Wenn dies abgeschlossen ist wird eine
6. XML-Datei generiert. Diese kann manuell um die Möglichkeit des

7. ApplicationLogin erweitert werden. Dies bedeutet, das der SecurityAspect später den Benutzer auch an der Wirtsapplikation anmelden kann. Außerdem besteht jetzt die Möglichkeit der manuellen Kontrolle der XML-Datei, bevor sie im nächsten Schritt

8. mit Hilfe des Datenbanktools in die Sicherheitsdatenbank eingebracht wird. Ab diesem Schritt ist der SecurityAspect in der Lage seine Aufgabe zu erfüllen und die Trainingsphase beendet.

9. Nun wird statt dem Trainingsaspekt der SecurityAspect mit der Applikation verwoben.

8.1 Trainingssuite

Die Trainingssuite dient der halbautomatischen Generierung der Beschreibung, in der alle Eigenschaften von Authentifikation und Autorisation definiert werden. Dazu gehören die Abhängigkeiten zwischen Benutzern, Rollen und Workflows sowie die Möglichkeit, für die Rollen die geforderten Authentifikationsmethoden anzugeben und bei den Benutzern die Authentifikationsmerkmale wie z.B. Passworte. Das Format dieser erzeugten Datei wird im Kapitel 8.2 genauer vorgestellt.

Die Suite besteht aus Webtool und Trainingsaspect. Der Aspect dient der Aufzeichnung und Übermittlung der Interaktion des Benutzers mit der Wirtsapplikation. Aus den aufgezeichneten Daten kann danach durch das Webtool die XML-Beschreibungsdatei erzeugt werden.

Der Trainingsaspect wird nachträglich in das zu schützende Programm hinzugefügt und fügt in eine Speicherstruktur bei jedem Seitenaufruf die übergebenen Parameter des Benutzers ein. Da diese Speicherstruktur durch beide Applikationen, also von der Wirtsapplikation und dem Webtool, gemeinsam verwendet werden soll, muss dies dem Tomcat explizit mitgeteilt werden. Dies passiert durch die crossContext-Option in der server.xml wie es in Listing 17 gezeigt wird.

```
<Context path="/web_tool" docBase="web_tool" reloadable="true" crossContext="true"/>

<Context path="/jpetstoreTraining" docBase="jpetstoreTraining" reloadable="true"
    crossContext="true" debug="10">
```

Listing 17: Änderungen an der server.xml

Ebenfalls muss die communicationInterface.jar im common/lib-Verzeichniss des Tomcat vorhanden sein. In dieser jar-Datei befindet sich die Beschreibung der gemeinsam verwendeten Speicherstruktur. Die Verwendung des Webtool wird in Kapitel 9.1 dargestellt.

In diesem Erzeugungsprozess können nur serielle Workflows entstehen, was durch die Verwendung der zu schützenden Applikation zum Erlernen der Workflows begründet ist, da sich der Zustand der Datenbank und Applikation ändert. Wenn einen Verzweigung im Ablauf des Workflows integriert werden soll, so würde dies bei einer Erzeugung durch das Webtool bedeuten, das die Applikation

in den Zustand dieses Punktes zurückversetzt werden muss. Dies kann durch ein nochmaliges Ausführen aller Interaktionschritte des Benutzers erfolgen und kann durch einen Aspect automatisiert werden. Allerdings befindet sich die Applikation danach nicht mit Sicherheit in dem gleichen gewünschten Punkt. Beispielsweise könnte ein Artikel nicht mehr ausreichend in der Datenbank vorhanden sein, so dass das Bestellmenü nicht erreicht werden kann, was aber vom Trainer gewollt war. Daher wurde das Problem der automatisierten Erzeugung von komplexen Workflows hier nicht weiter beleuchtet, eine Grundlage für die Erzeugung von komplexen Workflows ist aber gelegt. Eine mögliche Lösung kann in der Verwendung von gespeicherten Datenbankenzuständen zu jedem Zustand liegen. Allerdings erfordert diese Lösung eine weitaus komplexere Trainingssuite.

8.2 XML Beschreibung

Die durch die Trainingssuite erzeugte XML-Datei wird durch die Datei securityDescription.xsd definiert, welche der XML-Schemakonvention (vgl. [w3c04]) entspricht. Es wurde ein eigenes Datenformat entwickelt, welches eine effiziente Repräsentation der Workflows und den zugehörigen Autorisations- und Authentifikationsdaten ermöglicht. Der Einsatz eines bestehenden Formats, wie es in Form von SAML vorliegt, wurde verworfen, da dort keine einfache Repräsentation der Workflows möglich gewesen wäre. Somit wären zwei Beschreibungsdateien benötigt worden. Dies erschien als ein zu grosser Aufwand für den relativ begrenzten gewünschten Sprachumfang.

Die erzeugte XML-Datei entspricht einer Modellierungssprache die zwischen dem Generator und dem Interpreter Verwendung findet. Es besteht dadurch die manuelle Möglichkeit die Korrektheit der Beschreibung zu kontrollieren (Abbildung 27).

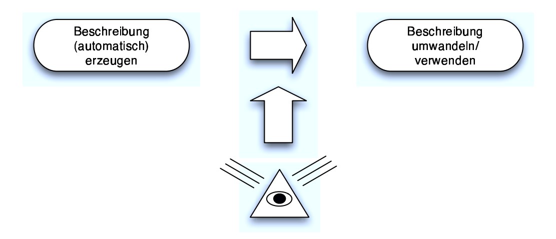


Abbildung 27: Verwendung der XML-Beschreibung

Der Aufbau unterteilt sich grob in vier Bereiche

- den Workflows
- den Benutzern
- den Rollen

- dem applikationsspezifischen Bereich

Durch die **Workflows** werden Arbeitsabläufe spezifiziert wie dies in Kapitel 4.3 beschrieben ist. Jeder Zustand der Statusmaschine wird beschrieben über einen Namen, die zugeordnete Javaklasse und den Übergängen zu anderen Zuständen. Dies wird durch Listing 18 ausgedrückt.

```
<xsd:complexType name="statesType" >
    <xsd:sequence>
        <xsd:element name="state" type="stateType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="stateType">
  <xsd:sequence>
    <xsd:element name="transitions" type="transitionsType" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="className" type="xsd:string"/>
</xsd:complexType>
```

Listing 18: Definition von Zuständen

Bei den Übergängen von einem Zustand zum Nächsten wird der Folgezustand angegeben, sowie die Bedingungen die eingehalten werden müssen. Für jeden Parameter kann ein regulärer Ausdruck oder ein SQL-Ausdruck oder beides angegeben werden. Der reguläre Ausdruck kann als Filter angesehen werden. Der SQL-Ausdruck produziert eine Liste, in der der Wert des Parameters enthalten sein muss. Für den SQL-Ausdruck werden zusätzlich noch diverse Parameter für den Datenbankzugriff angegeben. Die Schemadefinition eines Übergangs ist in Listing 19 gezeigt.

```
<xsd:complexType name="transitionType">
    <xsd:sequence>
        <xsd:element name="target" type="xsd:string"/>
        <xsd:element name="constraint" type="constraintType" minOccurs="0" maxOccurs="
            unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="constraintType">
    <xsd:sequence>
        <xsd:element name="para" type="xsd:string"/>
        <xsd:element name="regex" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="SQLConstraint" type="SQLConstraintType" minOccurs="0" maxOccurs="1
            "/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SQLConstraintType">
    <xsd:attribute name="sqlUser" type="xsd:string"/>
    <xsd:attribute name="sqlPassword" type="xsd:string"/>
    <xsd:attribute name="connectURL" type="xsd:string"/>
    <xsd:attribute name="sqlQuery" type="xsd:string"/>
</xsd:complexType>
```

Listing 19: Definition eines Übergangs

Ein **Benutzer** besteht aus einer Liste von Rollen, denen der Benutzer angehört und dem Benutzernamen (siehe Listing 20). Darüber hinaus werden noch die Authentifikationsparameter für den Benutzer eingetragen. Im Fall dieser Diplomarbeit sind ein Passwort sowie WibuKey als spezifische Parameter (siehe 5.3) aufgenommen worden. Auch sind alle Parameter zur Erzeugung einer Föderation (siehe 2.3) Bestandteil der Definition.

```
<xsd:complexType name="userType" >
    <xsd:sequence>
        <xsd:element name="role" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="idp" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="idpPassword" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="idpFirmCode" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="idpUserCode" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="applicationUser" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="applicationPassword" type="xsd:string" minOccurs="0" maxOccurs="1"
            />
        <xsd:element name="sp" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="idpNameIdentifier" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="spNameIdentifier" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

Listing 20: Definition eines Benutzers

Das Bindeglied von Workflows und Benutzern sind die **Rollen**. Eine Rolle besteht aus einem Namen und den Workflows, die zu dieser Rolle zusammengefasst werden.

Der Zusammenhang zwischen Benutzer, Rolle, Workflow und den Authentifikationsmethoden und Authentifikationsdaten wird in Abbildung 28 dargestellt.

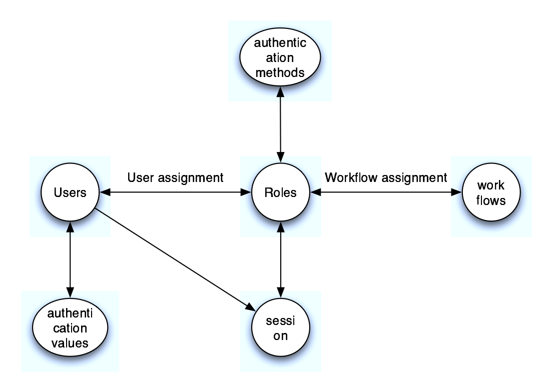


Abbildung 28: Zusammenhang von Usern, Rollen und Workflows

Im **applikationsspezifischem Bereich** ist hinterlegt, wo die Basisseite der Applikation zu finden ist. Hierzu wird die URL der Seite verwendet. Ausserdem finden sich dort die Parameternamen, in denen der Applikationsbenutzername und das dazugehörige Passwort vom Logindialog der Applikation erwartet werden.

8.3 Sicherheitsdatenbank

Der in 8.6 beschriebene Sicherheitsmonitor benötigt eine globale Datenquelle, auf deren Basis er seine Entscheidungen trifft. Bei der technischen Ausführung bestehen verschiedene Möglichkeiten, diese Quelle zur Verfügung zu stellen. Hierbei stellen sich zwei Wege zur Auswahl:

- Speichern der Workflows in XML-Dateien und Verwenden dieser Dateien als Basis oder

- Übertragen der XML-Dateien in ein relationales Schema und Speicherung in einer Datenbank.

Bei der ausschließlichen Verwendung von XML-Strukturen wird es nötig, für die Dauer einer Session ein Document Object Model (DOM) der Struktur im Speicher zu behalten oder bei jedem Zugriff die XML-Struktur neu auszuwerten. Dies ist entweder extrem speicherintensiv oder relativ langsam. Auch wäre der Zustand einer Session nur der Session bekannt. Erweiterte Zugriffsschutzregeln, wie sie z.B. Chinese Wall (siehe 4.2) vorsieht, wären in diesem Fall nicht zu implementieren. Es wurde daher die Verwaltung der Datenbasis an ein Datenbanksystem gegeben und die Hypersonic Database [hDG04] verwendet.

Hierdurch ist eine persistente Speicherung der Zustände für jeden Benutzer sowie ein effizienter Zugriff durch die Concurrency Control Mechanismen der Datenbank auch bei vielen gleichzeitigen Sessions auf dem Server gewährleistet. Hier sei auf die Literatur von Ramakrishnan und Gehrke [Ram98] sowie Bernstein et al. [BHG87] verwiesen.

Es werden im Rahmen des Demonstrators zwei getrennte Datenbanken neben den eventuell vorhandenen Datenbanken der Wirtsprogramme verwendet. Eine Datenbank wird vom IDP verwendet und liefert die Datenbasis der Authentifikation. Die zweite wird vom SecurityAspect bzw. dem Referenzmonitor verwendet und liefert alle Informationen, die der Authorisation, Reauthentifikation sowie der Anmeldung am Wirtssystem dienen. Diese Datenbanken sind getrennt, da sie getrennten Aufgaben dienen und auch auf weit entfernten Rechnern liegen. Desweiteren kann sich die Administration des IDP dem SP entziehen, so dass hier auch eine klare Trennung von Verantwortlichkeiten nötig ist.

Die verwendete Datenbankstruktur, auf die der Sicherheitsmonitor zurückgreift, wird in der Abbildung 29 dargestellt. Abbildung 30 zeigt die Tabelle, die verwendet wird, um die Informationen für eine Anmeldung eines Benutzers an der zu schützenden Applikation zu verwahren.

Alle angelegten Tabellen sind bis auf userState, sessionId und FSMState als nur-lesend zu verstehen. Somit teilt sich die Verwendung in die Repräsentation der User, Rollen und Workflows sowie in die persistente Speicherung des Zustandes jeder aktiven Session.

Ein Workflow wird in der Datenbank durch die Tabellen workflow, workflowStates und actionClass dargestellt. Jeder Workflow besitzt einen Namen und eine ID, zu der dann durch workflowStates die Zustände eines Workflows zugeordnet werden. Es existiert für jeden Constraint eines Übergangs (aus einem Zustand in einen Folgezustand) eine Zeile in der Tabelle.

Der aktuelle Zustand wird in der FSMState erfasst, in der zu jeder SessionId gespeichert wird, in welchem Zustand der jeweilige Workflow ist. Bei einem Zustandswechsel wird der neue Zustand vermerkt.

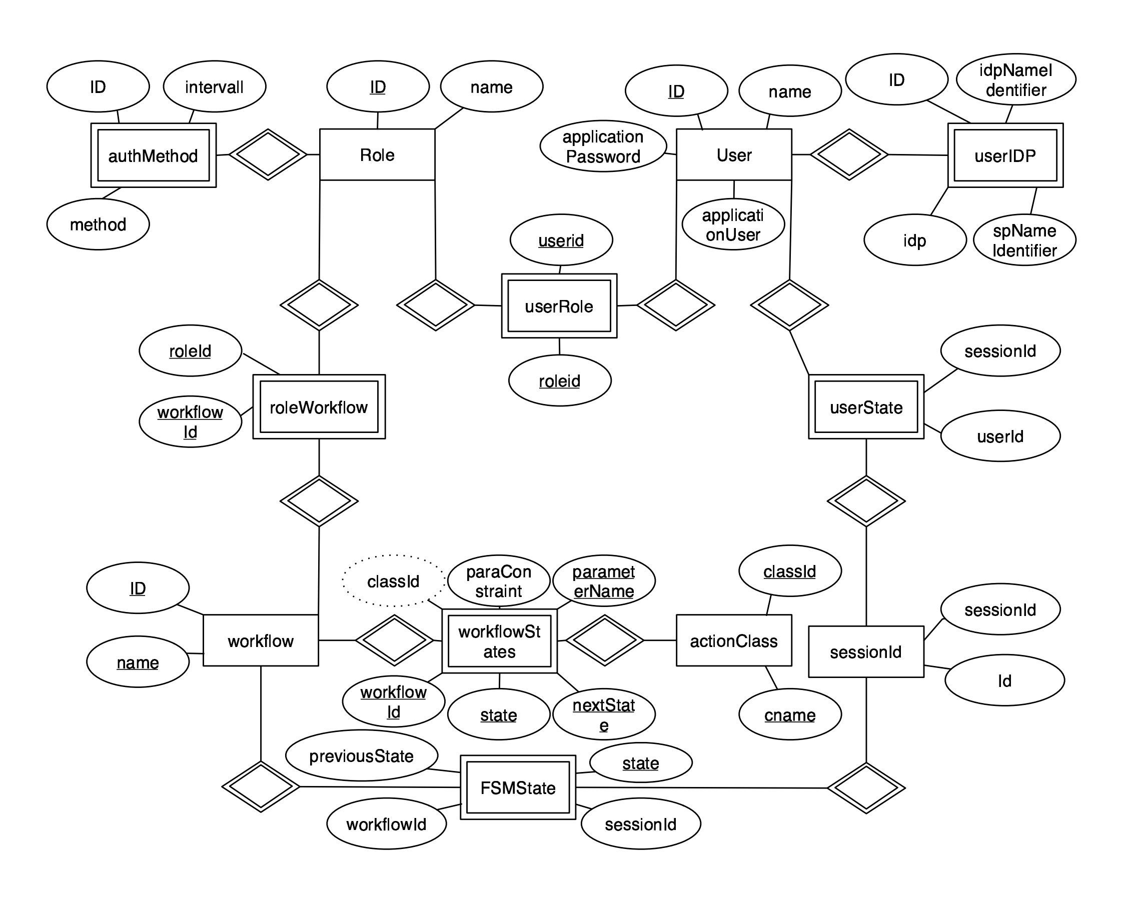


Abbildung 29: relationales Modell der Datenbank I

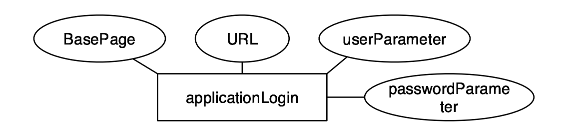


Abbildung 30: relationales Modell der Datenbank II

Der IDP verwendet die in Abbildung 31 gezeigte Struktur. In der accounts-Tabelle werden die Benutzer mit ihren Authentifikationsdaten erfasst. Die verschiedenen Federations, die ein Benutzer potentiell mit verschiedenen Service Providern haben kann, werden durch die Tabelle federations erfasst (vergl. Kapitel 2.3)

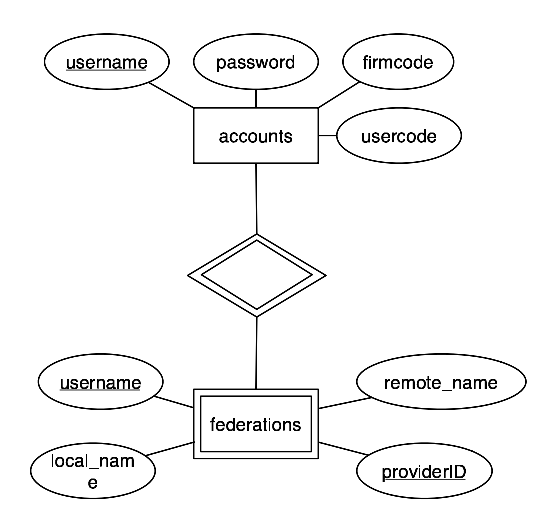


Abbildung 31: relationales Modell der IDP Datenbank

8.4 Datenbanktool

Das dbtool konvertiert die XML-Beschreibung in die relationale Repräsentation. Hierzu wird die XML-Datei serialisiert und mit Hilfe von SAX [Meg04] interpretiert. Eine weitere Funktion ist das Entfernen und Neuaufbauen aller existierenden Tabellen in der Datenbank.

8.5 SecurityAspect

Der SecurityAspect umhüllt wie in Abb. 32 gezielt die Methoden, welche vom Applikationserver aufgerufen werden und an die die Parameter der Interaktion des Benutzers übergeben werden. Ein solcher Parameter besteht immer aus einem Schlüsselnamen sowie einem zugehörigen Wert und stellt direkt die Eingaben des Benutzers an der Browseroberfläche dar. Diese Parameter stellen somit die Eingaben an die Applikation dar und verändern den Zustand der zu schützenden Applikation.

Der SecurityAspect erweitert den normalen Ablauf der Applikation um die Autorisation, Authentifikation, dem Anmelden des Benutzers bei der Applikation sowie die Möglichkeit, Autorisations- und Authentifikationsfehler abzufangen und

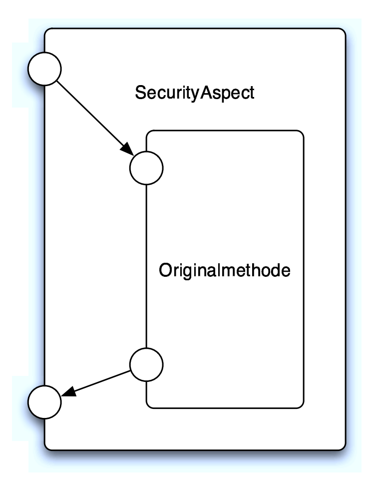


Abbildung 32: Der Securityaspect umhüllt die Methoden

dem Benutzer anzuzeigen. In einem begrenzten Rahmen kann der Benutzer auch auf einen Fehler reagieren

Die **Authentifikation** wird durch einen Aufruf des IDP realisiert wie in Listing 21 dargestellt wird. Falls noch kein Benutzer durch SourceID gesetzt wurde, wird ein Forward an die authnRequest Klasse durchgeführt (vgl. Kapitel 2.6). Diese führt dann eine Authentifikation des Benutzers, wie in Kapitel 2.3 beschrieben, durch.

```
user = (String)session.getAttribute("org.sourceid.sso.userID");
if (user == null){
  callPasswordIDP(session, request, response);
  return(null);
}
```

Listing 21: Aufruf der Authentifikation

Im Listing 22 werden verschiedene Attribute definiert und dann als authnRequest an den IDP weitergeleitet. Nach dem Zurückkommen sind verschiedene Werte im response-Object fest gesetzt, so dass es hier nötig ist, den Ablauf zu beenden und ein `return null` auszuführen. Dies hat zur Konsequenz, dass die aktuelle Seite wieder aufgerufen wird. Nun aber mit einem gesetzten Usernamen.

```
    private void callPasswordIDP(HttpSession session, HttpServletRequest request,
        HttpServletResponse response) throws Exception{
    Vector idpIdVector = getIDPVector(session);
    request.setAttribute("ProviderID", idpIdVector.firstElement());
    request.setAttribute("IsPassive", new Boolean(false));
    request.setAttribute("ForceAuthn", new Boolean(false));
    request.setAttribute("Federate", new Boolean(false));
    String myURL = request.getServletPath();
    request.setAttribute("Return.Success", myURL);
    request.setAttribute("Return.Failure", myURL);
    request.getRequestDispatcher("/sso/authnRequest").forward(request, response);
}
```

Listing 22: Aufruf der Authentifikation

Da im ersten Schritt nur eine Passwortauthentifikation durchgeführt wird, um den Benutzer festzustellen, ist es danach je nach Rollenkonfiguration nötig eine

Reauthentifikation durchzuführen. Eine solche Reauthentifikation kann auch periodisch durchgeführt werden. Hierdurch kann z.B. erzwungen werden, dass eine SmartCard während der Programmausführung vorhanden ist.

Die **Autorisierung** umkapselt den Aufruf der Methode. Dies wird aus dem Codebeispiel in Listing 23 deutlich. Bevor die Methode durch ein proceed (siehe 3.1.1) aufgerufen wird, stellt der SecurityAspect an den Referenzmonitor (siehe 8.6) die Anfrage, ob der Zugriff erlaubt ist. Falls der Zugriff nicht gewährt wird, erzeugt der Aspect eine Fehlerseite, die zwei Möglichkeiten zur Verfügung stellt:

- zurückzugehen auf die letzte Seite und eine andere Eingabe versuchen oder
- ganz auf die erste Seite zurückzugehen.

Das Zurückgehen wurde über ein Javascript realisiert, welches lediglich den back-Knopf des Browsers betätigt. Da dadurch aus dem Stack des Browsers eine gespeicherte Seite geholt wird und die Applikation sich nicht im Zustand verändert hat, kann der Benutzer eine neue Eingabe an den Server senden. Die Funktion des Backbuttons wird in [Abi04] und [Cor] erklärt. Um auf die erste Seite der Applikation zurückzugehen, ist es erforderlich, dies auch dem Referenzmonitor mitzuteilen, damit sich dieser auf die Ausgangsposition zurückbegeben kann. In der vorliegenden Implementation ist dies dadurch gelöst, dass die Workflows alle auf „Base" gestellt werden und ein Redirect auf eine in der Datenbank hinterlegte Seite durchgeführt wird, falls ein Parameter BackBasePage auf einen definierten Wert gesetzt ist. Durch den Redirect wird der Browser des Benutzers aufgefordert, eine neue Seite zu laden von einer frei definierbaren Quelle.

```
if (validate(thisJoinPointStaticPart.getSignature(),request,user)){
    ActionForward result = (proceed(mapping, form, request, response));
    return result;
}else{
    log.debug("around(): validated to false, page access denied");
    throwValidationErrorPage(response, request);
    return (null);
}
```

Listing 23: Kontrolle der Autorisation

Um den Benutzer vor einem doppelten Login zu bewahren, am IDP und an der Applikation, meldet der SecurityAspect den authentifizierten Benutzer an der Wirtsapplikation an. Hierzu ist in der Datenbank hinterlegt, welcher Benutzername und Passwort gegenüber der Applikation für diesen Benutzer verwendet werden muss. Desweiteren ist hinterlegt in welchen Parametern diese Daten übergeben werden müssen.

In Abbildung 33 wird die Abfolge der beschriebenen Prozesse dargestellt. Bei jedem Zugriff auf den Server durch den Benutzer wird zuerst der Referenzmonitor gestartet. Falls dies nicht möglich sein sollte und dieser einen Fehler übergibt, wird direkt der weitere Ablauf unterbunden und eine Fehlerseite erzeugt. Grund hierfür kann in der aktuellen Implementierung hauptsächlich ein Fehlen der Datenbank sein. Direkt darauf wird geprüft, ob der IDP einen Fehler zurückgegeben hat.

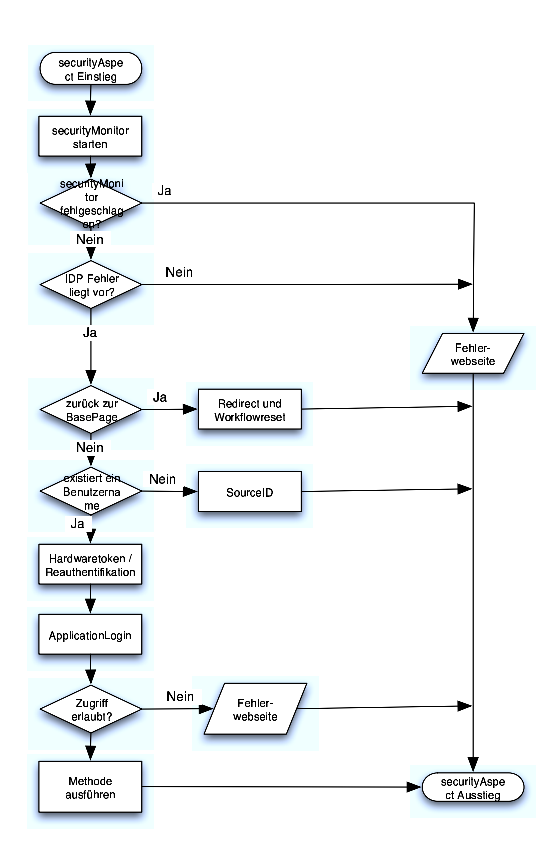


Abbildung 33: Flussdiagramm des securityAspect

Dies muss so früh wie möglich getestet werden, um eventuelle Endlosschleifen zu verhindern. Auch dies führt zu einer Fehlerseite, die dem Benutzer übermittelt wird.

Der hier vorliegende SecurityAspect ist für eine Verwendung in Verbindung mit dem struts-Framework gedacht. Als Weiterentwicklung ist es aber möglich ohne Struts auszukommen. Hierzu muss lediglich auf die Eigenschaften der Servletprogrammierung geachtet werden, die zu großen Teilen im Struts-Framework versteckt sind. Besonders wichtig ist hier, das die ankommenden Eingaben des Benutzers über verschiedene Methoden an die Applikation übergeben werden:

- doGet behandelt HTTP GET Anfragen. Die Daten die dem Server übermittelt werden, werden durch ein „?" getrennt an die URL angehängt.

- doPost empfängt HTTP POST Anfragen. Diese sind in ihrer Länge nicht begrenzt und finden ihre Anwendung zum Beispiel in der Übertragung von Kreditkarteninformationen. Die Daten werden im Gegensatz zu GET für den Benutzer unsichtbar an den Server übermittelt.

- doPut ist ähnlich einer FTP-Übertragung und erlaubt das Transferieren von Dateien auf den Server.

Da Struts eine sehr hohe Verbreitung hat (siehe z.B. [Hai03]) und durch die Verwendung des Frameworks diese Details der Programmierung von Webservices unsichtbar werden, haben wir uns entschieden, im Demonstrator Struts einzusetzen. Ein weiterer Vorteil ergibt sich auch aus dem Vorhandensein des JPetStore in Struts, der als Standardbeispiel in der Webprogrammierung gilt.

8.6 Referenzmonitor

Der durch den SecurityAspect verwendete Referenzmonitor befindet sich in der Klasse SecurityMonitorHistory. Wie der Name schon andeutet, entscheidet dieser Referenzmonitor auf der Basis der bereits getätigten Interaktionen eines Anwenders mit der Software und verbietet jede Handlung, die nicht explizit erlaubt ist. Als Entscheidungsbasis wird die in 8.3 beschriebene Sicherheitsdatenbank verwendet, in der lediglich Positivregeln hinterlegt sind. Bei einem Zugriff wird nach einer Regel gesucht, die diesen erlaubt. Wenn keine Positivregel existiert, wird der Zugriff verwehrt. Dies bildet die Basis eines Konzepts der minimalen Rechte wie es z.B. in [FK92] gefordert wird. Auf der Entscheidungsbasis wird das in Kapitel 4.2 beschriebene rollenbasierte Sicherheitsmodell (RBAC) verwendet.

Dementsprechend wird bei jeder Interaktion des Benutzers mit dem System auf Basis des Benutzernamens und der SessionID ermittelt

- in welchen Rollen der Benutzer Mitglied ist und daraus die Menge der Workflows gebildet,

- aus den Rollen wird auch die Menge der benötigten Authentifikationsmerkmale hergeleitet und

- der Zustand der einzelnen Workflows bestimmt.

Rollen, Benutzer, Sessions und Workflows hängen wie in Abbildung 28 gezeigt zusammen. Die SessionID wird vom Tomcat vergeben und als Identifikator verwendet, um den aktuellen Zustand der Workflows einer Session zu speichern. Die SessionID ist notwendig, da ein Benutzer durchaus auch mehrere Sessions besitzen kann. Den Rollen sind die geforderten Verfahren zur Authentifikation zugeordnet. Ein Benutzer, der diese Anforderungen nicht erfüllen kann, da er z.B. keine Smartcard besitzt, aber diese gefordert ist, ist nicht in der Lage sich zu authentifizieren.

Somit besitzt ein Benutzer die Menge aller zu seiner Rolle gehörenden Rechte. In diesem Modell kann er auch gleichzeitig in verschiedenen Workflows aktiv sein. Falls durch einen Interaktionsschritt ein Workflow nicht weitergeführt werden kann, so wird der Workflow auf „Base" gesetzt. Falls alle Workflows auf „Base" stehen bedeutet dies, dass es keinen Workflow gibt, auf dem der Benutzer weiterarbeiten kann und der Referenzmonitor validiert zu false.

Die gleichzeitige Aktivität von Workflows bedingt sich aus dem Aufbau von Webservices. Zumeist gibt es eine zentrale Seite, von der aus die Software und die möglichen Abläufe erreichbar sind. Somit kommt diese Seite in allen möglichen Workflows vor. Da es aber nicht entscheidbar ist, welchen Schritt der Benutzer gehen wird, ist somit jeder Workflow, der dem Benutzer möglich ist, zu diesem Zeitpunkt aktiv. Dies sollte sich allerdings nach wenigen Schritten bereits geändert haben und nur noch wenige bzw. einer der Workflows sollte sich herauskristallisieren.

Um nun dem Benutzer die Möglichkeit zu geben, auf eine durch ihn erzeugte Fehleingabe zu reagieren (er könnte ja unabsichtlich in einen verbotenen Bereich gegangen sein), wird im FSMState der jeweils letzte Zustand der Workflows hinterlegt. Falls nun alle Workflows in den Zustand „Base" übergegangen sind, so wird ein Schritt zurückgegangen. Der SecurityAspect bekommt damit die Möglichkeit eine Fehlerseite anzuzeigen und durch Betätigen des Backbuttons kommt der Benutzer zu der letzten Maske zurück, ohne dass sich der Zustand der Applikation verändert hat.

Durch diese starke Eingrenzung der Funktionalität wird der eigentliche Code des Referenzmonitors relativ kurz und somit auch gut kontrollierbar. Dies wird durch Listing 24 demonstriert, welches den kompletten Funktionsumfang der Methode isValidAccess beinhaltet. Wie beschrieben werden hier im ersten Schritt die Rollen des Benutzers bestimmt, um dann in nächsten Schritt die Workflows zu ermitteln. Nun wird jeder Workflow weitergeschrieben, also in einen Folgezustand geführt oder auf „Base", falls es aufgrund der Eingabe des Benutzers keinen erlaubten Folgezustand gibt. Die neuen Zustände werden danach in die Datenbank

geschrieben. Aufgrund des neuen Datenbankzustands wird erst am Ende entschieden, ob der gewünschte neue Zustand erreicht werden kann. Falls alle Workflows im Zustand „Base” stehen, wird der Zustand abgelehnt und ein „false” als Antwort gegeben.

```
try{
        Hashtable newFSMStates = new Hashtable();
        Vector roles = database.getRoles(user);
        String sessionId = request.getRequestedSessionId();
                for(Enumeration rolenames = roles.elements(); rolenames.hasMoreElements();){
                String rolename = (String)rolenames.nextElement();
                for(Enumeration workflows = database.getWorkflows(rolename).elements();
                    workflows.hasMoreElements();)
                {
                        String workflow = (String)workflows.nextElement();
                        FSMState state;
                        try{
                                state = database.getActualWorkflowState(sessionId, workflow);
                        }catch(SQLException e){
                                database.updateFSMState(sessionId, workflow, "base");
                                state = database.getActualWorkflowState(sessionId, workflow);
                        }
                        String nextState = state.allowedTransition(className+"."+methodName,
                            request);
                        if (nextState==null) nextState= new String("base");
                        newFSMStates.put(workflow, nextState);
                        }
                }
        for(Enumeration en = newFSMStates.keys(); en.hasMoreElements();){
                String key = (String)en.nextElement();
                database.updateFSMState(sessionId,key,(String)newFSMStates.get(key));
        }
        return database.existsValidWorkflow(sessionId);
} catch(SQLException e){
        return false;
}
```

Listing 24: Auszug aus dem securityMonitor

8.7 HtmlRewriterAspect

Der HtmlRewriterAspect hat folgende Aufgaben:

- Darstellen der erzeugten Debugmeldungen des SecurityAspect und des IDP. Dies macht es für den Entwickler überflüssig, in einem getrennten Programm die erzeugten Meldungen zu untersuchen.
- Einbindung des Applets zur Realisierung der Proaktivität.

Um diese Ziele zu erreichen wurde der Buildprozess wie in 3.4 beschrieben verwendet. Dadurch wird es möglich jede JSP-Seite nachträglich zu verändern. Erreicht wird dies durch eine Kapselung des verwendeten Responseobjekts, welches im Aspect der eigentlichen Methode übergeben wird. Dies wird in Listing 25 gezeigt.

```
        WrappedResponse wrappedResponse = new WrappedResponse(response, request);
        proceed(request, (HttpServletResponse)wrappedResponse);
```

Listing 25: Änderungen an der web.xml

Sobald die Ausführung beendet wird erfolgt das Umschreiben der erzeugten HTML-Seite. Dazu wird lediglich eine Methode der gekapselten Klasse aufgerufen. Innerhalb dieser Methode wird dann eine Ersetzung der Tags „<body>” und „</bo-

dy>" durchgeführt durch die eine Tabelle eingefügt wird, deren Einträge entsprechend gesetzt werden. Bei der Ersetzung der Tags muss beachtet werden, dass diese Tags case-sensitive sind und es auch Parameter gibt, die erhalten bleiben müssen. Daher gestaltet sich der Ersetzungsbefehl relativ aufwendig, wie es in Listing 26 zu sehen ist.

```
input = input.replaceFirst("<([bB][oO][dD][yY])(.*)>","<$1 $2><table border=\"1\" width
     =\"100%\"><tr><td colspan=\"2\">");
```

Listing 26: Änderungen an der web.xml

Eine wichtige Einschränkung dieses Verfahrens ist, dass es darauf vertraut, dass der vorhandene HTML-Code korrekter Code ist. Im Fall des JPetStore z.B. wurde der body-Tag nicht geschlossen, so dass anfangs die Tabelle nicht korrekt eingefügt wurde. Es wäre für einen absolut generischen Einsatz dieses Aspects ein HTML-Parser einzubinden, der in der Lage ist, solche Fälle zu erkennen und den HTML-Aufbau in eine korrekte Form zu bringen.

8.8 Identity Provider

Der Identity Provider (IDP) hat die Aufgabe die Authentifikation von Subjekten durchzuführen und von ihm abhängige Module (Service Provider) mit relevanten Daten über die stattgefundenen Authentifikationen zu versorgen. Die Anforderungen im Rahmen dieses Demonstrators an den Identity Provider sind dabei eine initiale Authentifikation zur Übermittlung von Benutzerdaten und folgende periodische Reauthentifikationen.

Das Zusammenarbeiten von unterschiedlichen Service und Identity Providern wird hier mit Hilfe des Liberty Protokolls (Kapitel 2.2) durchgeführt. Als Framework zum Zugriff auf das Liberty Protokoll wird die Implementierung von SourceID (Kapitel 2.6) verwendet. Zur Gestaltung der Benutzerinteraktion wird Struts [Fou04b] verwendet. Da das Framework von SourceID nur die Kommunikation mit dem Liberty Protokoll abdeckt, mussten die Authentifikationsverfahren und passenden Webseiten selbst entwickelt werden. Die Behandlung einer Authentifikationaufforderung wird entweder durch direktes Aufrufen einer Webseite durch einen Benutzer oder durch Initiierung von einem beteiligten Service Provider gestartet. Der Ablauf einer von Benutzer direkt initiierten Authentifikation ist in Bild 34 dargestellt. Der Benutzer wird mit der Startseite des IDP begrüßt und bekommt die Auswahl, ob er eine Authentifikation per Passwort oder Wibu-Key wünscht. Falls er Passwort Authentifikation angeklickt hat, wird er zur ActionClass IndexPwAction.java weitergeleitet. Daraufhin bekommt der Benutzer eine Eingabemaske präsentiert und kann seinen Benutzernamen und entsprechendes Passwort eingeben. Nach der Übermittlung und Überprüfung der Daten wird dem SourceID Framework mittgeteilt, dass sich der User erfolgreich authentifiziert hat. Im Falle eine Authentifikation mittels Wibu-Key wird erst überprüft, ob sich der Benutzer schon per Benutzernamen und Passwort authen-

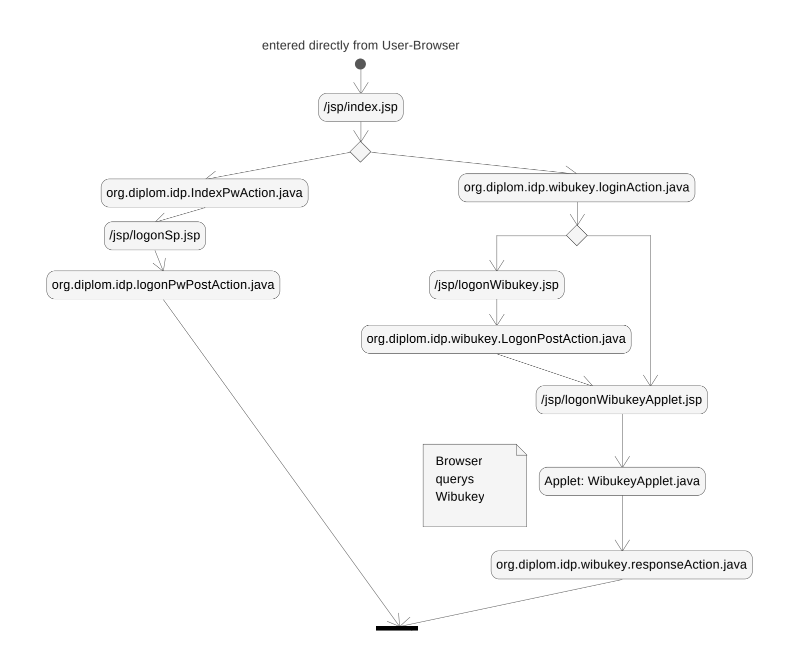


Abbildung 34: IDP Ablauf

tifiziert hat und im negativen Fall eine Eingabemaske für Benutzernamen und Passwort gesendet. Nach erfolgreicher Eingabe wird der Browser des Benutzers auf die Seite /jsp/logonWibukeyApplet.jsp geleitet, in der das Applet zur Wibu-Key-Authentifikation eingebettet ist. Diesem Applet werden der benutzerabhängige Firmcode und Usercode des Wibu-Keys übermittelt sowie die zufällig erzeugte Challenge und der Salt-Wert. (Seite 112). Die Antwort des Applet folgt an die Action-Class responseAction.java. Diese vergleicht die Response des Applets mit dem eigenerzeugten Ergebnis und informiert danach das SourceID Framework.

Eine Authentifikation, die durch einen Service Provider initiiert wurde, läuft unterschiedlich ab (Siehe Abbildung 35). Da eine Authentifikationsaufforderung von einem Service Provider über das Liberty Protokoll an den Identity Provider versendet wird, landet die Anfrage erst beim SourceID Framework. Dieses leitet nach Überprüfung der Aufforderung an die in den Konfigurationsdateien eingetragene Seite des Identity Providers weiter. Diese Seite entscheidet je nach Anforderung, ob eine Authentifikation per Passwort oder Wibu-Key stattfindet. Die Webseiten sind ähnlich gestaltet wie Webseiten der direkt initiierten Authentifikation. Beim Abschluss der Authentifikation wird die Kontrolle zurück an das SourceID Framework übergeben, welches den authentifizierten Benutzernamen und die Authentifikationsmethode an den Service Provider über das Liberty Protokoll zurück.

Zusätzlich zu den benötigten Einstellungen, die unter Kapitel 2.6 beschrieben sind, müssen für die Funktion des Identity Providers mit SourceID bei einer vom Service Provider initiierten Authentifikationsaufforderung noch weitere Ein-

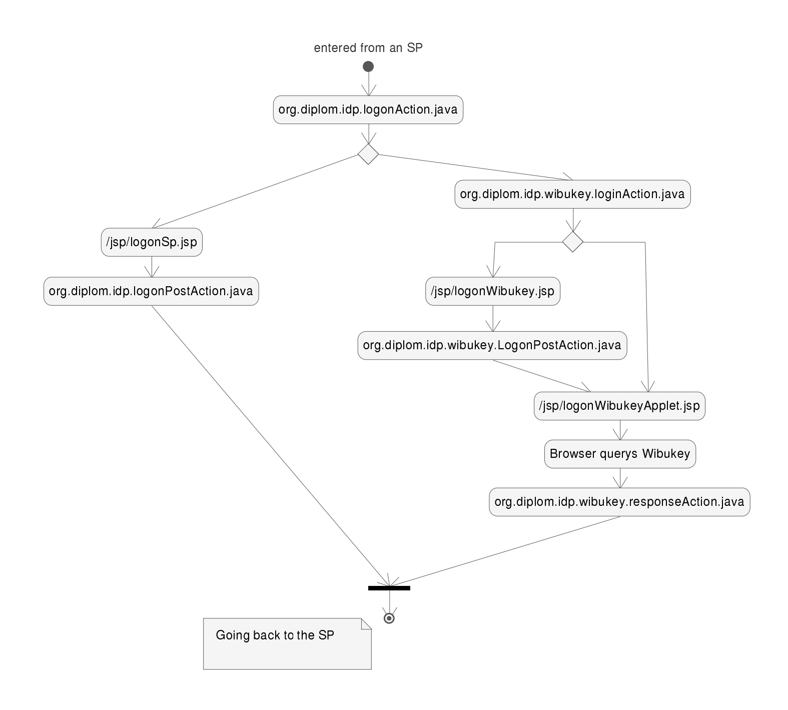


Abbildung 35: vom SP initiierter IDP Ablauf

stellugen getätigt werden. Da SourceID nur für die Kommunikation des Liberty Protokolls zuständig ist, aber bei einer Authentifikationsaufforderung die Kontrolle besitzt, muss SourceID über seine Konfigurationsdateien mitgeteilt werden, wo sich die Webseiten zur Abwicklung der Authentifikation befinden. Zu diesen Seiten wird der Browser des Benutzers von SourceID weitergeleitet.

Eine der wichtigsten Einstellungen ist die Angabe der <idp-authentication-uri> (siehe Listing 27). Dies ist der Einspringpunkt zur Applikation, wenn eine Authentifikation von einem Service Provider gewünscht wird.

```
<idp-authentication-uri>/logon_from_sp.do</idp-authentication-uri>
```

Listing 27: Auszug aus der sourceid-sso.xml

Weitere Einspringpunkte zur Abwicklung des Single Logout Verfahren sowie Aufbau und Beendigung einer Föderation (siehe Kapitel 2.3) werden ebenfalls in dieser xml-Datei spezifiziert.

8.8.1 Kanalverschlüsselung

Zur Durchführung einer Authentifikation, die von einem Service Provider initiiert wurde, müssen viele Nachrichten ausgetauscht werden. Zum einen gehören dazu die Nachrichten, die durch die Web Redirects des Liberty Protokolls zwischen den Service Providern und den Identity Providern entstehen. Da diese über den Browser des Client Rechners verschickt werden, ergibt sich eine Verdoppelung

des Kommunikationsaufkommens, da die Nachrichten vom Service Provider zum Client Rechner und von dort weiter zum Identity Provider geschickt werden. Eine genaue Beschreibung des Web Redirect Verfahrens befindet sich in Kapitel 2.4. Weiterhin gehören dazu die Nachrichten, die durch die Übertragungen der Webseiten vom Identity Providers entstehen. Unter der Vielzahl dieser Übertragungen sind folgende die schützenswertesten:

- Benutzernamen/Passwortes
- WibukeyApplets mit seinen Parametern
- Response vom WibukeyApplets

Einen guten Überblick über die nötigen Übertragungen bietet die Abbildung 36. Hier sind beispielhaft die abgebildet, die durch eine Authentifikation mit dem Wibu-Key stattfinden. Wie in Kapitel 2.5 angedeutet gibt es gewisse Problematiken bei der Durchführung der Web Redirect Technik des Liberty Protokolls. In [Gro03] wird empfohlen, einen sicheren Übertragungskanal für die Liberty Nachrichten zu verwenden. Eine Möglichkeit, den Übertragungskanal der Liberty Nachrichten abzusichern, ist die Benutzung von SSL (siehe Kapitel 2.5). Nach [Eck01, Kapitel 12.3.4] ist SSL ein de facto Standard für sichere HTTP-Verbindungen.

Die Authentifikation der Kommunikationspartner wird im Demonstrator dazu verwendet, dass sich der Identity Provider dem Clientrechner gegenüber ausweist. Hiermit kann der Benutzer sicher sein, mit welchen Servern er in Interaktion steht. Eine zusätzliche Authentifizierung des Client Rechners über SSL wird im Rahmen dieser Diplomarbeit nicht betrachtet. Dies könnte eine weitere unter vielen Möglichkeiten sein, die Authentizität eines Benutzers im Rahmen eines Identity Providers festzustellen.

Liberty Alliance schreibt in seiner Spezifikation „Liberty Id-FF Bindings and Profile Specification" [CK03] vor, dass bei dem Wunsch nach Nachrichtenintegrität und Vertraulichkeit SSL Version 3.0 verwendet werden muss (vgl. Kapitel 2.5). Die Intergration von SSL für die Liberty Protokoll Nachrichten ist mit Hilfe von SourceID realisierbar. Es muss bei einer Applikation dafür gesorgt werden, dass die Einspringpunkte von SourceID des Liberty Protokolls ausschließlich per SSL erreichbar sind. Somit hat man die Vertraulichkeit und Integrität der Nachrichten durch SSL sichergestellt. Dies muss natürlich bei allen Beteiligten des Circle of Trust geschehen, da sonst z.B. die Nachrichten nur vom Provider A zum Client Rechner per SSL abgesichert sind und weiter zum Provider B nicht per SSL abgesichert sind und somit die Vertraulichkeit und Integrität auf der zweiten Strecke nicht gegeben ist. Außerdem muss beachtet werden, dass die Nachrichten zwar auf dem Weg vertraulich und integer sind, aber beim Client-Browser umgepackt werden müssen und somit hier offen sind. Da wir für diesen Demonstrator selbstsignierte Zertifikate verwenden und SourceID bei Übertragung der

Nachrichten per SSL eine Überprüfung des SSL Zertifikat vornimmt, konnten wir den Austausch der Liberty Nachrichten nicht per SSL absichern. Bei Verwendung von nicht selbst zertifizierten Zertifikaten, deren Zertifizierungsstelle in der Java Zertifikatsdatenbank eingetragen ist, sollte dieses Problem beseitigt sein.

Zur Absicherung der oben genannten Übertragungen zwischen dem Identity Provider und dem Client-Rechner wird in dem Demonstrator SSL verwendet. Zu Anfang einer Kommunikation zwischen einem Client-Rechner und dem Identity Provider wird die Authentifikationsmöglichkeit von SSL verwendet. Dies ermöglicht dem Benutzer am Client-Rechner die Identität des Identity Providers zu überprüfen, damit er nicht fälschlicherweise seine Authentifikationsdaten an Dritte übermittelt. Weiterhin ist durch die Kanalabsicherung zwischen dem Identity Provider und dem Client-Rechner die Vertraulichkeit und Integrität der Nachrichten gewährleistet. Somit ist es einem Angreifer nur erschwert möglich, die Datenpakete zwischen dem Client-Rechner und dem Identity Provider zu entschlüsseln und an die Authentifikationsdaten des Benutzers zu kommen.

Um am Tomcat-Server die Benutzung von SSL zu aktivieren, müssen mehrere Vorbereitungen getroffen werden. Als erstes muss in der Datei **server.xml** der Connector für SSL aktiviert werden. Dies wird durch das Einfügen des in Listing 28 gezeigten Connectors realisiert.

```
<Connector port="8443"
           maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
           enableLookups="false" disableUploadTimeout="true"
           acceptCount="100" debug="0" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS" keystoreFile=".keystore"/>
```

Listing 28: Änderungen an der server.xml

Wichtig dabei ist die Angabe des Ports und des keystoreFile. Als Port wurde im Demonstrator nicht der SSL Standardport 443 verwendet sondern der Port 8443. Die Angabe des keystoreFile zeigt auf die Datei, die die zur SSL-Authentifikation verwendeten Schlüsselpaare enthält. Hier wurde ein selbstsigniertes Zertifikat verwendet. Anleitungen zum keystoreFile sind in [Hor04] und [Pro04] zu finden. Mit dem Hinzufügen dieses Connectors ist es prinzipiell möglich alle Kommunikation zwischen einem Client-Rechner und einem Tomcat-Server sowohl über das normale HTTP-Protokoll als auch über SSL abgesichertes HTTP-Protokoll abzuwickeln.

Um sicherzustellen, dass die Übertragung der Webseiten des Identity Providers per SSL abgesichert sind, wird bei dem Demonstrator sslext [SSL04] verwendet. Dies ist eine Erweiterung des Struts Frameworks zur Spezifikation des Übertragungsprotokolls. Hiermit kann angegeben werden, welche Webseite bzw. Action-Class über welches Protokoll übertragen werden soll. Somit kann während des Ablaufs einer Webapplikation zwischen reiner HTTP-Übertragung und SSL abgesicherter Übertragung gewechselt werden. Hiermit ist es möglich, nicht wichtige Seiten ohne SSL Absicherung übertragen zu lassen und wichtige Seiten nur mit Absicherung durch SSL übertragen zu lassen. Bei dem Identity Provider unseres Demonstrators wird die Einstiegsseite per normalem HTTP übertragen

und sobald es zur Übermittlung von Authentifikationsdaten kommt wird zu SSL gewechselt. sslext übernimmt dabei nicht nur die Kontrolle, ob Action-Klassen und JSP Seiten mit dem richtigen Protokoll aufgerufen werden, sondern auch die Spezifikation der Links je nachdem, ob ein Wechsel zwischen SSL abgesichertem und normalem HTTP Protokoll stattfindet. Dies ist nötig, da die Links normalerweise nur relative Links in Relation zum Pfad der Webapplikation des Tomcat sind und beim Wechsel zwischen SSL und HTTP direkt die Angabe einer vollständigen URL nötig ist.

Die Integration von sslext in Struts wird durch das Austauschen des action-mappings-Parameter in der Datei **struts-config.xml** wie in Listing 29 dargestellt realisiert.

```
<action-mappings type="org.apache.struts.config.SecureActionConfig">
```

Listing 29: Änderungen an der struts-config.xml

Um genau spezifizieren zu können, welche JSP-Seiten nur per SSL erreichbar sind, wird der Parameter wie in Listing 30 zu jeder abzusichernden JSP-Seite eingetragen.

```
<sslext:pageScheme secure="true" />
```

Listing 30: Änderungen an einer JSP-Seite

Die Angabe, ob der Aufruf einer Action-Class nur per SSL oder direkt erfolgen darf, geschieht in der Datei **struts-config.xml**. Wie in Listing 31 gezeigt, gibt es dazu einen Parameter secure, der Wahlweise auf true oder false gesetzt werden kann.

```
<action
    path="/indexPw"
    type="org.diplom.idp.IndexPwAction"
    scope="request">
        <set-property property="secure" value="true"/>
        <forward name="success" path="/jsp/logon.jsp" redirect="true"/>
</action>
```

Listing 31: Änderungen an der struts-config.xml

8.9 Wibu-Key

Der Wibu-Key kann im Rahmen dieses Demonstrators neben der initialen Authentifikation zur periodischen Reauthentifikation (siehe 5.2.2 auf Seite 65) des Benutzers verwendet werden. Nachdem der Benutzer initial seine Benutzerkennung und sein Passwort angegeben hat, kann periodisch eine Überprüfung auf Vorhandensein des Wibu-Keys am Client Rechner durchgeführt werden. Die Angabe, welches Authentifikationsverfahren durchgeführt werden soll, wird vom Service Provider spezifiziert. Im Falle unseres Demonstrators wird einer Rolle ein Authentifikationsverfahren zugeordnet. Die Überprüfung des Wibu-Keys wird mit Hilfe eines Applets realisiert. Dieses Applet wird vom Identity Provider geholt und lokal auf dem Client-Rechner innerhalb des Browsers ausgeführt.

Das **Authentifikationsverfahren** mit dem Wibu-Key wird nach dem im Kapitel 5.3 beschriebenen Verfahren durchgeführt. Dem Applet werden die Parameter Firm-Code, User-Code, Salt und RAND mitgegeben. Der Salt Wert wird dem Wibu-Key als Selection-Code in den Verschlüsselungsalgorithmus übergeben und dient dazu, die Zufälligkeit zu erhöhen. Der RAND Parameter entspricht den Daten, die mit Hilfe des Keys verschlüsselt werden sollen. Das aus der Verschlüsselung mit dem Wibu-Key erhaltene Ergebnis schickt das Applet über eine Weiterleitung des Browsers an die Klasse responseAction.java des Identity Providers zurück. Ein Beispielablauf ist in Abbildung 36 dargestellt und wird im folgenden

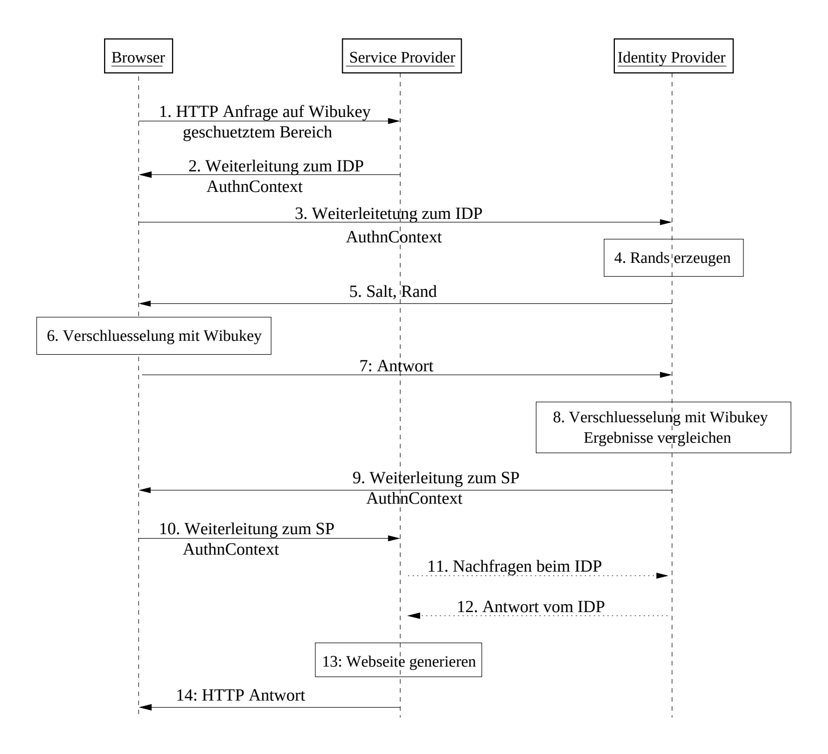


Abbildung 36: Ablauf Wibukey

kurz erläutert.

Der Benutzer will mit seinem Browser auf eine Webseite eines Service Providers zugreifen (1). Der Service Provider initiiert eine Authentifikationaufforderung mit dem Wibu-Key und leitet den Browser (2, 3) des Benutzers weiter auf die Webseiten des Identity Providers. Dieser erzeugt die RAND und den Salt-Wert (4). Daraufhin bekommt der Client-Rechner das Applet mit den Parametern User-Code und Firm-Code, passend zu dem aktuellen Benutzer, sowie Salt und RAND Wert geschickt (5). Das Applet nimmt Kommunikation mit dem Wibu-Key auf und verschlüsselt den RAND Wert mit den Angaben Salt, User-Code und Firm-Code als Schlüssel (6) (siehe Kapitel 5.3). Das Ergebnis wird über eine Weiterleitung des Browsers zu der responseAction Klasse des Identity Providers geschickt (7). Der Identity Provider überprüft (8) das Ergebnis des Applets mit

dem am Server angeschlossenen Wibu-Key und leitet den Browser des Benutzers mitsamt dem Ergebnis der Authentifikation über letztere Weiterleitung an den Service Provider zurück (9, 10). Der Service Provider generiert nach positiv verlaufender Authentifikation die gewünschten Webseiten für den Benutzer (13, 14). Es ist darauf hinzuweisen, dass das gerade beschriebene Verfahren von den Autoren entwickelt wurde und vor einem produktiven Einsatz außerhalb des Demonstrators einer genauen Sicherheitsanalyse zu unterziehen ist (vgl. Kapitel 10).

Da die benötigten Daten zur Verschlüsselung bzw. das Ergebnis über das Internet übertragen werden ist es sinnvoll, diese Daten mit einer Kanalverschlüsselung abzusichern (siehe Seite 107), damit kein Angreifer in den Besitz von Klartext/Kryptotext-Paare kommt und einen known-plaintext Angriff durchführen kann.

Die Integration des Wibu-Keys im Identity Provider wird im Prinzip über zwei Webseiten bzw. zwei Action-Classes getätigt. Die erste Action-Class loginAction.java generiert die beiden zufälligen Werte. Diese beiden Werte und die Parametern User-Code und Firm-Code werden der JSP-Seite logonWibukeyApplet.jsp übergeben, in der das Applet eingebettet ist. Das Applet löst nach dem Verschlüsseln der RAND eine Weiterleitung zu der responseAction.java aus und übergibt dieser das Ergebnis. Hier wird auf Seiten des Servers die API des Wibu-Keys mit den gleichen Parametern aufgerufen und die beiden Ergebnisse verglichen. Falls das Ergebnis positiv ist, wird ein AuthnContext Object (siehe Kapitel 2.4) erzeugt und der Service Provider mit diesem Object über die Art der erfolgten Authentifikation benachrichtigt.

8.9.1 WibukeyApplet

Das Applet **WibukeyApplet.java** ist wie schon beschrieben in die JSP-Seite logonWibukeyApplet.jsp eingebettet. Das Applet bekommt als Parameter alle nötigen Informationen übergeben. Das Übertragen des Applets bzw. der JSP-Seite wird nur mittels Kanalabsicherung SSL vom Identity Provider zugelassen. Das Applet wird außerdem signiert, um einerseits die benötigten größeren Sicherheitsfreigaben für das Applet von der Java Sandbox zu bekommen als auch es dem Benutzer zu ermöglichen, eine Überprüfung durchführen zu können, ob das Applet von dem richtigen Server geholt wurde. Zusätzlich werden dadurch die Parameter, die für das Applet benötigt werden, nicht im Klartext übertragen.

In Abbildung 37 ist zusammengefasst, welche Kommunikation nötig ist, um eine Authentifikation mittels Wibu-Key durchführen zu können. Zwischen Identity Provider und dem Applet werden die benötigten Daten mittels SSL abgesichert. Das Applet greift auf der Client Seite über die API auf den Wibu-Key zu. Die API allerdings greift nicht direkt auf den Wibu-Key zu sondern auf einen Wibu-Key

Server. Dieser Server erledigt die Hardware Kommunikation mit dem Wibu-Key und ist über einen TCP Port erreichbar. Hier wäre zu klären, in wieweit dieser Port des Wibu-Key Servers nach außen geöffnet ist, um nicht Angreifern von außerhalb die Möglichkeit zu geben, Angriffe auf den Wibu-Key Server direkt tätigen zu können. Auf Seiten des Servers geschieht eine ähnliche Kommunikation. Die Actionclass responseAction.java greift auf die Java-API des Wibu-Keys zu, um die kontrollierende Wibu-Key Verschlüsselung durchzuführen. Auch hier auf Seiten des Servers wird auf den Wibu-Key mittels eines Wibu-Key Servers zugegriffen.

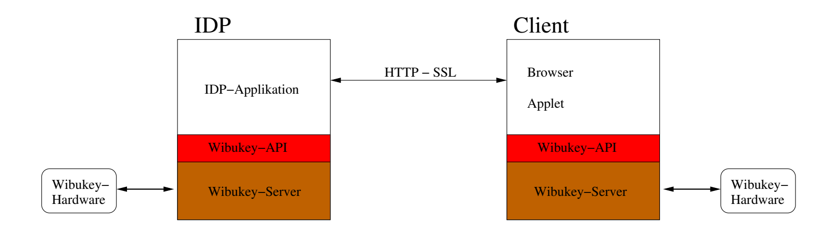


Abbildung 37: Wibukey

8.10 ProAktivApplet

Proaktivität wird in diesem Demonstrator innerhalb eines Applets verwendet. Das Applet läuft auf Seiten des Client-Rechners und dient zur Überwachung des Benuters und der periodischen zero-interaction (siehe Kapitel 5.2.2) Reauthentifikation. Nach einer initialen Authentifikation mit Benutzername und Passwort werden weitere Reauthentifikationen ohne Interaktion des Benutzers durch das Überprüfen des Wibu-Keys durchgeführt. Hieraus ergibt sich die Gefahr einer möglichen Vergesslichkeit beim Benutzer. Er könnte das Vorhandensein des Wibu-Keys am Rechner vergessen und somit Angreifern z.B. ermöglichen innerhalb einer Mittagspause, bei der der Benutzer seinen Computer verlässt, Zugang zu dem System durch diesen Rechner zu bekommen (vgl. Kapitel 5.2.2).

Um genau dieses Problem zu adressieren, haben wir in unserem Demonstrator ein proaktives Applet eingebaut. Dieses Applet läuft auf dem Client-Rechner des Benutzers und überprüft das Vorhandensein des Wibu-Keys im System. Falls der Wibu-Key vorhanden ist, der Benutzer im System eingeloggt ist und der Benutzer eine gewisse Zeitspanne inaktiv war, bekommt er eine Warnung und der Identity Provider wird informiert, dass der Benutzer die letzte Zeit inaktiv war. Somit könnte der Benutzer vom Identity Provider z.B. automatisch ausgeloggt werden oder Vergesslichkeitslisten für jeden Benutzer erstellt und Maßnahmen zur Schulung dieser Benutzer ergriffen werden. Bei der Realisierung dieses proaktiven Applets sind wir auf Probleme mit dem Konzept des Liberty ID-FF Protokolls (siehe Kapitel 2.3) gestoßen. Ein Kontakt zwischen dem Client Rechner und dem

Identity Provider besteht beim Single-Sign On Konzept des Liberty ID-FF Protokolls nur zu Authentifikationszwecken. Somit ist dem Identity Provider zwar bekannt, wann zuletzt eine Reauthentifikation stattgefunden hat, ihm ist aber nicht bekannt, ob der Benutzer in letzter Zeit aktiv war oder nicht. Dies kann nur ein Service Provider wissen, da der Benutzer nur direkt mit den Seiten des Service Providers arbeitet. Um nun sowohl den Service Provider als auch den Identity Provider zu integrieren wird das proaktive Applet beim Aufbau jeder Seite des Service Providers durch den HtmlRewriterAspect (siehe 8.7) gestartet. Jede Seite, die vom Service Provider, d.h. im Falle unseres Demonstrators von JPetstore, generiert wird, wird vom HtmlRewriterAspect vor Übertragung an den Clientrechner verändert. In den Quelltext wird ein Link auf das Applet bei dem Identity Provider gesetzt. Der Grund für das Nachladen des Applet direkt vom Identity Provider liegt darin, dass über diesen Weg eine einfache Applet-Server Kommunikation zum IDP aufgebaut werden kann. Bei Übertragung einer zweiten Webseite vom Service Provider wird das alte Applet von der alten Seite automatisch vom Browser beendet und das gleiche Applet mit der zweiten Seite erneut gestartet. Ein Screenshot des Applet, eingebettet in eine Beispielseite des JPetstores, ist in Abbildung 54 oben links zu sehen. Der Kreis nimmt dabei die Farbe grün an, wenn der Wibu-Key vorhanden ist und das Zeitlimit noch nicht überschritten ist. Falls der Wibu-Key nach Zeitüberschreitung noch vorhanden ist, wechselt die Farbe auf Rot. Die periodische Kommunikation zwischen Applet und Identity Provider findet dabei auch nach der Zeitüberschreitung weiterhin statt. Man könnte damit z.B. eine Statistik über die Länge der Abwesenheit eines Benutzers erstellen.

Trotz des ständigen Neuladens des Applets existiert außerdem eine kontinuierliche Kommunikation zwischen dem Applet und dem Identity Provider, da die Kommunikation zwischen beiden zustandslos ist. Auf Seiten des Identity Providers steht das Servlet **ProAktivAction.java** mit dem Applet in bidirektionaler Kommunikation. Sowohl das Laden des Applets als auch der Nachrichtenaustausch zwischen Applet und Identity Provider werden dabei über SSL abgesichert. Da das Applet zum gleichen Zeitpunkt gestartet wird wie die letzte Seite aufgebaut wird, hat das Applet den Zeitpunkt der letzten Interaktion des Benutzers zur Verfügung. Eingaben des Benutzers zwischen den Seitenübertragungen werden bei unserem Demonstrator nicht berücksichtigt, könnten aber in das Applet zur genaueren Interaktionskontrolle zusätzlich eingebaut werden.

Die Kontrolle über den Aktivitätsstatus eines Benutzers muss dabei auf Seiten des Identity Providers sein, da ein Benutzer an mehreren Service Providern gleichzeitig angemeldet sein und arbeiten kann. Nur der Identity Provider hat genauen Überblick, an welchen Service Providern der Benutzer arbeitet und wie lange die Lebensdauer der Authentication Assertions für jeden Service Provider ist.

Als Zeitspanne wird bei unserem Demonstrator die Lebensdauer einer vom Identity Provider ausgestellten Authentication Assertion genommen. Diese Zeit-

spanne wird wie in Listing 32 in der Konfigurationsdatei von SourceID auf Seiten des Identity Providers in der Datei **sourceid-sso.xml** spezifiziert.

```
<idp-authn-lifespan>1800</idp-authn-lifespan>
```

Listing 32: Auszug aus der sourceid-sso.xml

9 Beispielablauf des Demonstrators

Im folgenden wird die Verwendung der in Kapitel 8 vorgestellten Software demonstriert. Der Ablauf gliedert sich in die bereits beschriebenen Einheiten:

- Training
- Editieren der XML-Beschreibung
- Erzeugung der Sicherheitsdatenbank
- Einsatz des Systems

Die Durchführung wird exemplarisch am JPetstore von iBatis gezeigt. Die Applikation wird hierbei nicht verändert[3].

9.1 Erzeugen der Sicherheitsbeschreibung

Um die Traingsphase durchführen zu können, muss der TrainingAspect mit der Web-Applikation kompiliert werden. Dazu reicht es aus, die Datei TrainingAspect.-java in das src-Verzeichnis der Applikation zu kopieren und die Applikation zu kompilieren. Außerdem muss das Build-Script des dbtools ausgeführt werden, damit die zur Kommunikation zwischen dem TrainigsAspect und Webtool benötigten Klassen zur Verfügung stehen. Dieses Script erzeugt die jar-Datei securityDatabase.jar und kopiert diese ins Verzeichnis common/lib des Tomcats, damit die beiden Applikationen eine gemeinse Kommunikationsbasis aufbauen können. Da der Trainingsaspect in einer anderen Anwendungsumgebung als das Web Tool läuft, muss der Tomcat-Server so konfiguriert werden, dass er eine Kommunikation zwischen beiden gestattet. Dies wird durch den in Listing 33 gezeigten Parameter crossContext in der Datei server.xml des Tomcats erreicht.

```
<Context path="/web_tool" docBase="web_tool" reloadable="true" crossContext="true"/>
<Context path="/jpetstore" docBase="jpetstore" reloadable="true" crossContext="true"
    debug="10">
```

Listing 33: Änderung an der server.xml

Um einen Workflow aufnehmen zu können, muss zuerst im Webtool, unter dem Punkt **workflow generator**, die Aufzeichnung eines Workflows gestartet werden (Abbildung 38). Daraufhin kann man in der Applikation beginnen einen Workflow aufzuzeichnen. Hierbei kann die Applikation ganz normal benutzt werden und es werden die benötigten Daten vom TrainingAspect aufgezeichnet. Beispielhaft wurde ein Workflow beim JPetstore aufgezeichnet. Abbildung 39 zeigt die Startseite des JPetstores. Danach wurde ein Fish angeklickt und man gelangt zur Seite in Abbildung 40. Daraufhin wurde auf ein Goldfish wie in Abbildung 41

[3] Allerdings befindet sich in der Datei IncludeBottom.jsp des JPetStore ein Fehler. Dort fehlt ein schließender body-Tag. Ohne diesen ist die erzeugte HTML Seite nicht korrekt und kann nicht vom HTMLRewriter verändert werden. Dieser Fehler fällt im Normalfall nicht auf, da die Browser am Markt diese Tags selbstständig ergänzen.



Abbildung 38: Starten einer Workflowaufzeichnung

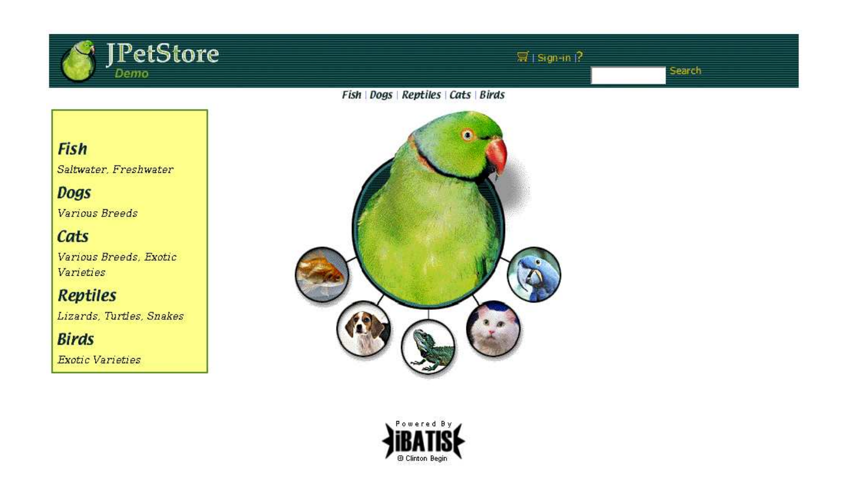


Abbildung 39: Startseite des JPetstores

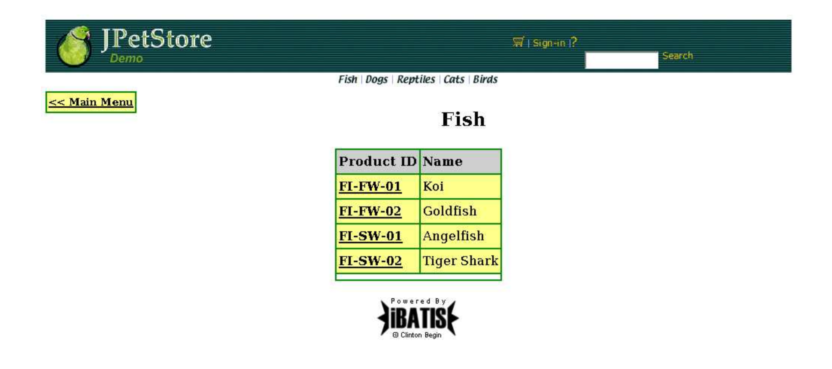


Abbildung 40: JPetstore - FISH

angewählt. Die Aufzeichnung kann beliebig viele weitere Webseiten enthalten und wird hier aus Platzgründen nicht weiter geführt. Das Beenden der Aufzeichnung



Abbildung 41: JPetstore - Goldfish

eines Workflows kann im Webtool mit dem Button „Save Recorded Workflow" getätigt werden (Abbildung 42).



Abbildung 42: Beenden einer Workflowaufzeichnung

Zur Bearbeitung eines aufgezeichneten Workflows geht man oben auf den Punkt **workflow editor**. Daraufhin werden alle bisher aufgezeichneten Workflows angezeigt und man wählt einen aus (Abbildung 43). Der Workflow wird



Abbildung 43: Bearbeitung der Workflows

daraufhin als Ablaufdiagramm angezeigt (Abbildung 44). Jeder der Knoten entspricht einer aufgezeichneten Actionclass, in unserem Falle State genannt, und es ist möglich diese States anzuklicken, um die Parameter angezeigt zu bekommen und eventuell zu bearbeiten. Beispielhaft wird der zweite State angeklickt,

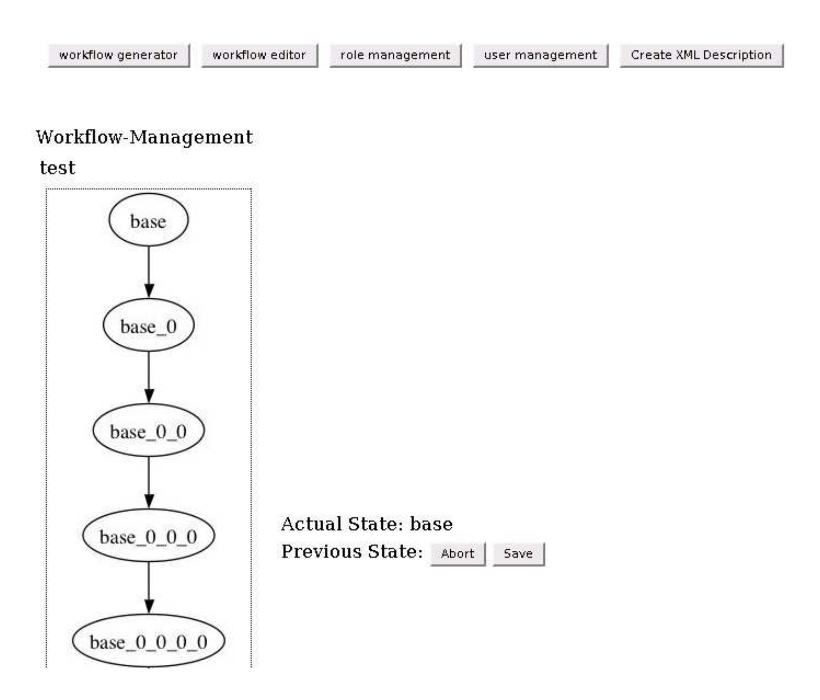


Abbildung 44: Bearbeitung eines Workflows

Abbildung 40, und die passenden Parameter werden angezeigt (Abbildung 45). Hier sind die Parameter zu erkennen, die der ActionClass übergeben wurden. In diesem Fall „FISH" und andere Parameter. In dieser Eingabemaske ist es möglich, eine SQL Anfrage zu spezifizieren, aus der der SecurityAspect den Vergleich für den entsprechenden Parameter tätigen kann.

Wenn mindestens ein Workflow angelegt wurde, können Rollen unter dem Punkt **role managment** angelegt werden (Abbildung 46). Nachdem Rollen angelegt wurden, können diese bearbeitet werden (Abbildung 47), wobei spezifiert werden kann, welche Workflows dieser Rolle zugeordnet sind und welche Authentifikationsmethoden sie erfordern (Abbildung 48). Nachdem mindestens eine Rolle angelegt wurde, können Benutzer unter dem Punkt **user management** angelegt werden (Abbildung 49). Daraufhin bekommt man eine Liste der Benutzer angezeigt (Abbildung 50). In der Bearbeitungsmaske des Benutzers (Abbildung 51) wird ein Benutzer einer Rolle zugeordnet. Außerdem wird der Name des Identity Providers, über den sich der Benutzer einloggt, angegeben. Zusätzlich wird der Benutzername und das Passwort für das einloggen bei der abzusichernden Applikation angegeben. Das IDP-Passwort, der SC-Firmcode und der SC-Usercode dienen zum Erzeugen der IDP-Datenbank, falls der IDP des Demonstrators verwendet wird.

Nachdem alle benötigten Daten im Webtool eingegeben sind, kann unter dem Punkt **Create XML Description** eine XML-Beschreibung erzeugt werden, die alle aufgenommenen Workflows und die gespeicherten Daten enthält (Abbildung 52).

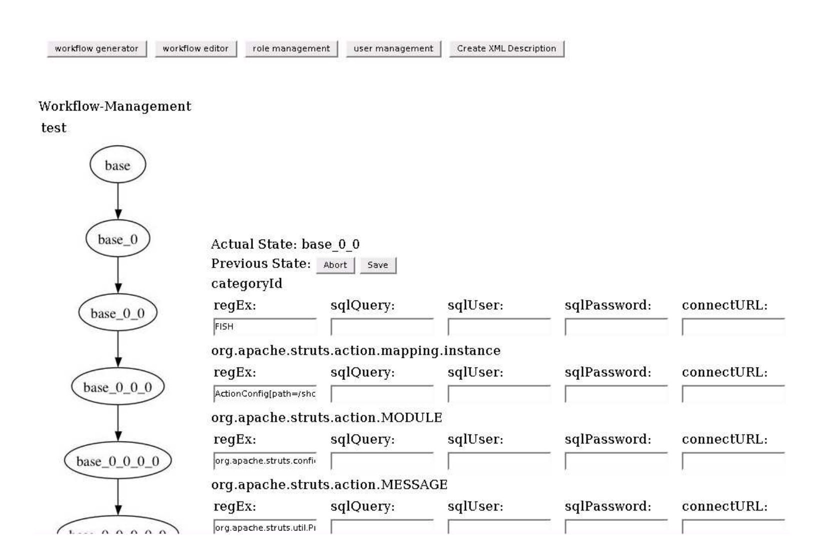


Abbildung 45: Bearbeitung eines States

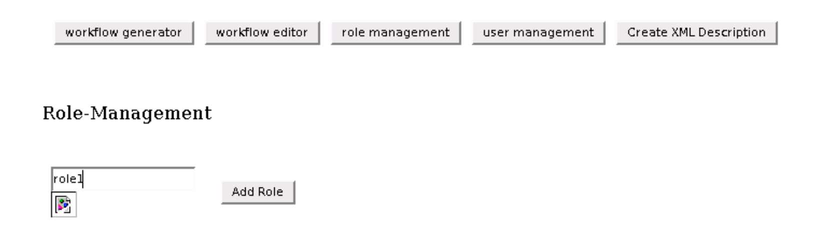


Abbildung 46: role management

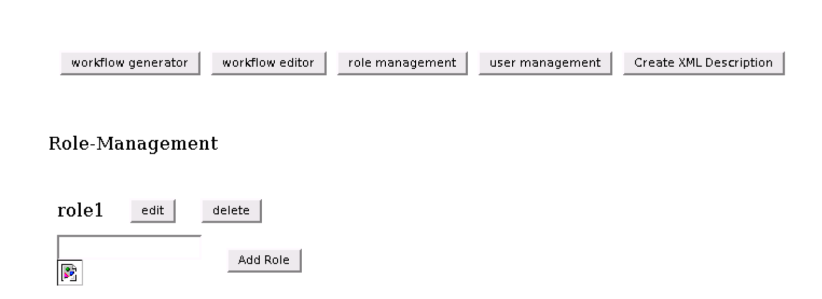


Abbildung 47: Übersicht der Rollen

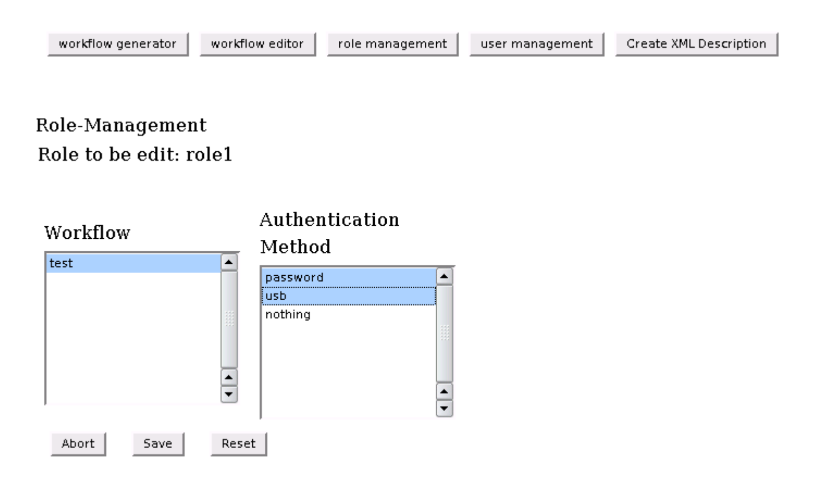


Abbildung 48: Rollenbearbeitung

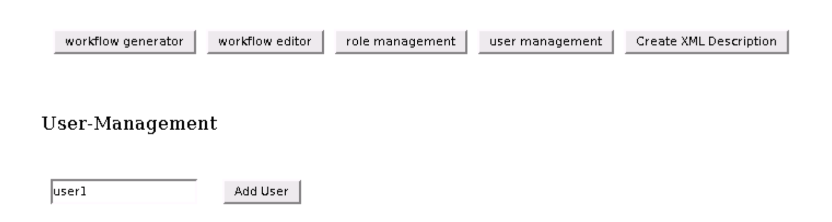


Abbildung 49: User Management

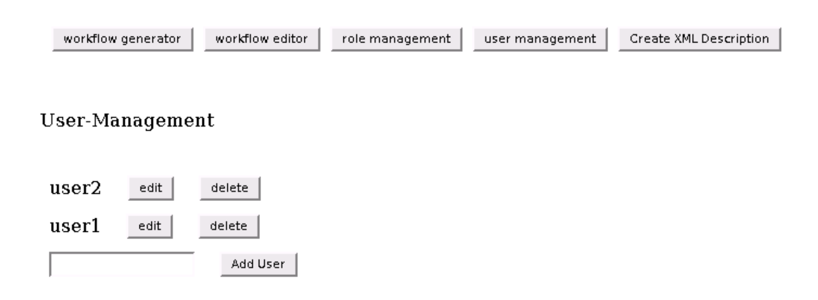


Abbildung 50: Übersicht der Benutzer

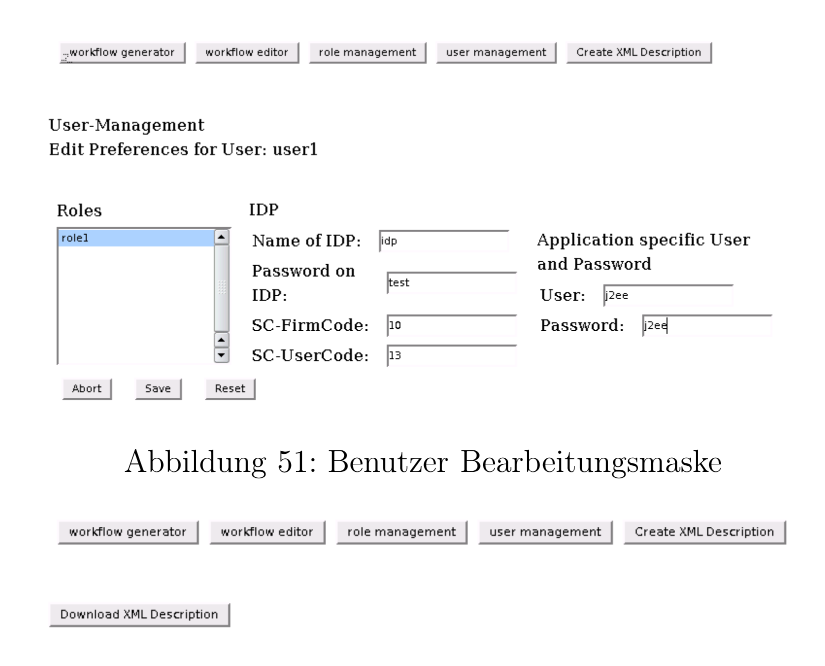


Abbildung 51: Benutzer Bearbeitungsmaske

Abbildung 52: Erzeugung der XML-Beschreibung

9.2 Manuelles Bearbeiten der Beschreibung

Nachdem im Webtool die Beschreibung erzeugt wurde, kann sie manuell nachbearbeitet werden. Dieses Beispiel beschränkt sich auf die in Listing 34 gezeigte Zeile. Durch sie wird die Basisseite der Workflows festgelegt. In den Parametern url, userParameter und passwordParameter sind die lokale Loginseite und die Namen der Parameter hinterlegt, in denen der Benutzername und das dazugehörige Passwort übergeben wird.

```
<applicationLogin url="/shop/signon.shtml" userParameter="username" passwordParameter="
    password" basePage="/jpetstore/shop/index.shtml"/>
```

Listing 34: Änderung an der xml-Datei

9.3 Konvertierung in eine relationale Datenbank

Die Konvertierung der XML-Beschreibung in eine relationale Datenbank findet durch das Tool **db tool** statt. Hierzu muss die Sicherheitsdatenbank gestartet sein und mittels des Programms db_tool.sh, mit der XML-Beschreibung als Parameter, wird die Sicherheitsdatenbank aufgebaut.

9.4 Produktiveinsatz

Nachdem die Sicherheitsdatenbank erstellt wurde, muss die abzusichernde Web-Applikation, in dem Fall unseres Demonstrators der JPetstore, erweitert werden.

Hierzu gehören die Einstellungen, die für die Einbindung des Liberty-Protokoll mittels SourceID getätigt werden müssen, sowie die Einbindung der Aspecte HTMLRewriter und SecurityAspect. Die Details werden im folgenden erläutert.

Einbinden von SourceID Für die Konfiguration von SourceID müssen die drei Dateien web.xml, sourceid-sso.xml und sourceid-sso-providers.xml geändert werden. Die nötigen Änderungen sind in Kapitel 2.6 beschrieben. Zusätzlich benötigt SourceID einige jar-Dateien im lib-Verzeichnis der Applikation. Diese sind:

- axis.jar
- castor-0.9.4.2-xml.jar
- commons-discovery.jar
- commons-httpclient.jar
- commons-logging.jar
- jaxrpc.jar
- saaj.jar
- sourceid-sso.jar
- xml-apis.jar
- xmlsec.jar

Da jeder Provider in einem Circle of Trust ein eigenes Zertifikat für die Signierung von Liberty Nachrichten benötigt, muß für SourceID ein solches Zertifikat erstellt werden. Dieses Zertifikat wird in einer Datei gespeichert und der Dateiname in die Datei sourceid-sso.xml eingetragen.

HTMLRewriter Zur Einbindung des HTMLRewriters reicht es, das Build-Script des HTMLRewriters auszuführen. Das Build-Script führt dabei zuerst eine Kompilation der Dateien HTMLRewriterAspect.java und WrappedResponse.java durch. Diese beiden Dateien werden in das Archiv HTMLRewriterWeave.jar gepackt und in das lib-Verzeichnis der Applikation kopiert. Zusätzlich werden die, vom Tomcat erzeugten Java-Klassen der JSP-Seiten gelöscht. Dies geschieht, um eine Neukompilation der JSP-Seiten vom Tomcat zu erzwingen, sodass zu jeder Datei von Tomcat automatisch der HTMLRewriter-Aspect hinzugefügt wird.

SecurityAspect Für die Einbindung des Security-Aspects muss zuerst das Build-Script ausgeführt werden. Dieses kompiliert die zum Zugriff auf die Datenbank benötigte Java Dateien und packt diese in das Archiv securityAspect.jar. Dieses Archiv wird ich das lib-Verzeichnis der Applikation kopiert. Zusätzlich wird der Aspect SecurityAspect.java in das src-Verzeichnis der Applikation kopiert. Nach einer Kompilation der Webapplikation ist der SecurityAspect mit allen Action-Classes der Applikation verwoben.

Beispielablauf Nach Aufruf des, mit dem SecurityAspect abgesicherten JPetstores, wird der Benutzer zuerst zu den Seiten des Identity Providers weitergeleitet und bekommt die Login-Seite des IDPs (Abbildung 53) angezeigt. Falls das Login-

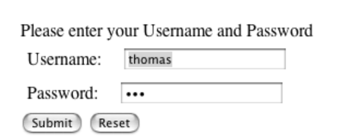


Abbildung 53: Identity Provider - Login Page

Formular richtig ausgefüllt wurde, wird die Wibu-Key Authentifikation durchgeführt und der Benutzer zurück zum JPetstore geleitet. Daraufhin bekommt der Benutzer die Ausgabe des JPetstores, abgeändert durch den HTMLRewriter Aspect, angezeigt (Abbildung 54). Hier ist das ProAktiv Applet auf der linken

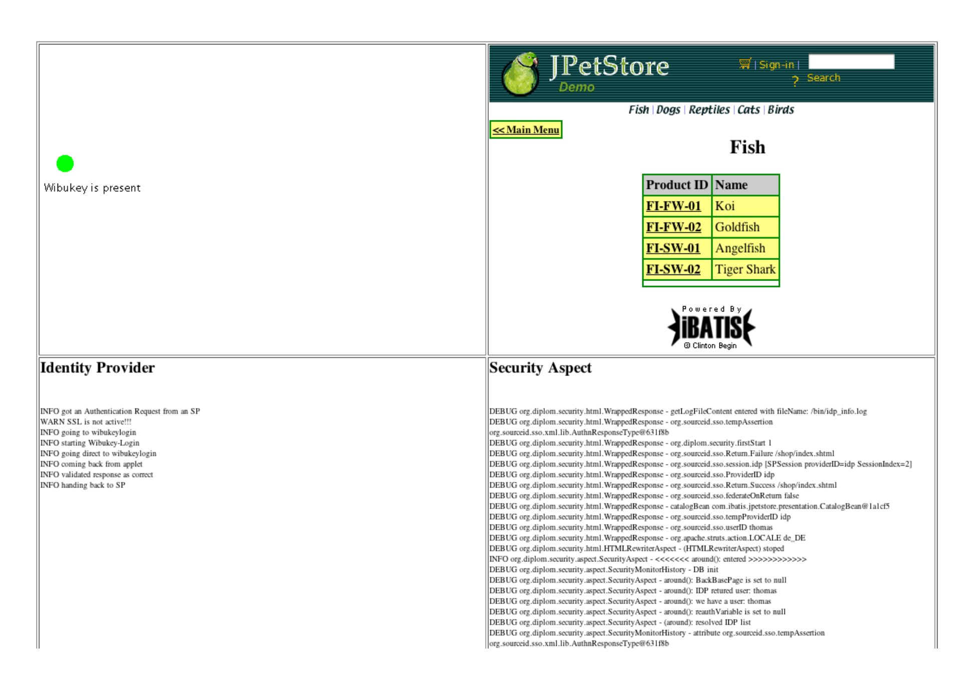


Abbildung 54: Produktiveinsatz

Seite zu sehen. Dieses Applet ist im Normalfall grün. Wenn der Benutzer längere Zeit inaktiv gewesen ist wechselt das Applet auf die Farbe rot (Abbildung 55). Falls der Benutzer im weiteren Verlauf den Workflow verläßt und auf eine nicht

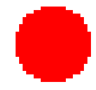


Abbildung 55: ProAktiv Applet

erlaubte Seite zugreifen will, bekommt er stattdessen eine Fehlerseite angezeigt (Abbildung 56). Daraufhin kann er zur letzten Seite zurückkehren, um den Work-

You are not allowed to be there ....
Go back
Back to Base Page

Abbildung 56: Fehlerseite

flow fortzusetzen oder kann auf die Startseite wechseln, um einen neuen Workflow anzufangen.

10 Fazit und Ausblick

Das in dieser Diplomarbeit vorgestellte Konzept hat es ermöglicht zu einer Applikation eine neue Funktionalität auf einfache Weise hinzuzufügen. Durch diese hinzugefügte Funktionalität erhält das System eine neue Qualität, da eine flexible und dynamische Authentifikation und Autorisation zur Verfügung steht. Das Liberty-Framework als Basis der Authentifikation ermöglicht neue Wege in der Kooperation unterschiedlicher Firmen, da die Integration externer Mitarbeiter schnell und effizient realisierbar ist. In Verbindung mit der vorgestellten flexiblen Autorisation der Benutzerinteraktion können die gewünschten Arbeitsabläufe effizient beschrieben werden. Somit ist eine Kooperation von Firmen in kurzer Zeit etablierbar. Darüber hinaus erfolgt durch den möglichen Wiedereinsatz des Sicherheitsmoduls eine Komplexitätsreduktion der Wirtsapplikation. Hierdurch ist sie billiger und schneller in der Entwicklung.

Die in Kapitel 8 vorgestellte Implementierung hat verdeutlicht, dass es möglich ist, Sicherheit nachträglich in einen verteilten Webservice einzuführen. Die klare Trennung des Benutzungsinterface von der Programmlogik und die einheitlichen Schnittstellen dazwischen ermöglichen den in dieser Diplomarbeit verfolgten Weg. Ohne diese Schnittstellen ist ein Einfügen von Workflows nicht so einfach möglich. Daher ist es schwer zu beurteilen, inwieweit das vorgestellte Konzept sich außerhalb von Webservices verwenden lässt. Ein alleiniges Hinzufügen von Authentifikationsmethoden ist aber im Prinzip wie in Kapitel 3.2.2 gezeigt möglich.

Da der Demonstrator Sicherheit nachträglich in ein System integriert, ist die Betrachtung der Sicherheit der modularen Sicherheitsfunktionen und ihrer Kommunikation ein wichtiger Punkt. Die genaue Betrachtung möglicher Angriffe und Sicherheitsprobleme wird in Kapitel 7 durchgeführt. Hierbei hat sich heraus gestellt, dass der Identity Provider in diesem Konzept eine mögliche Schwachstelle darstellen kann. Da durch einen denial-of-Service Angriff auf diesen Identity Provider auch alle angeschlossenen Service Provider außer Gefecht gesetzt sind, ist in großen Systemen unbedingt auf Redundanz bei den Identity Providern zu achten. Das Problem des Session Hijackings durch das Stehlen von Cookies vom Browser des Benutzers ist ein weiteres mögliches großes Problem.

Modularität Es ist aber auch deutlich geworden, dass die Trennung von Programm und Sicherheit Probleme erzeugt, da diese getrennt arbeiten. Eine Einschränkung der Art „Verkaufe keine Hunde, wenn keine Hunde im Lager sind" ist nicht formulierbar. Diese Grenze wurde versucht aufzuweichen, indem der Datenbankzustand der Wirtsapplikation mit in Betracht gezogen wird. Allerdings fehlt hierbei noch eine bessere Formalisierung der Beschreibungssprache. Momentan ist es nur möglich zu kontrollieren, ob sich ein Parameter exakt in der Ergebnismenge einer SQL-Anfrage befindet. Dies ist zu wenig, als dass damit bereits Sicherheitspolicies formuliert werden könnten.

Proaktivität Die Proaktivität wird durch ein Applet erreicht, welches in jede erzeugte Webseite eingebettet wird. Bei jeder Webseite muss der Browser das Applet beenden und neu starten. Dies erzeugt unnötigen Datentransfer, da bei jeder Übertragung einer Webseite auch das Applet übertragen werden muss. Falls der Browser die Java-Applets zwischenspeichert, fällt die Verzögerung aufgrund der Übertragungszeiten wesentlich geringer aus. Daneben wird für das Beenden und Neustarten Rechenleistung auf der Clientseite verbraucht. Diese Probleme müssen bei einer Implementierung des Systems in stark frequentierten Systemen in Betracht gezogen werden.

Da ein Benutzer gleichzeitig oder nacheinander mit Webseiten mehrerer Service Provider arbeiten kann (die Applikation kann z.B. zum Lastausgleich auf mehrere Server verteilt sein) muss das Applet in jede Seite die generiert wird eingebettet werden. Nur dann kann kontinuierlich überprüft werden, ob der Benutzer an seinem PC ist.

Die Einbindung des Applets in den HTML-Quellcode jeder erzeugten Webseite ist nur eine Möglichkeit, eine kontinuierliche Überwachung des Benutzers zu erreichen. Falls es möglich ist, ein Programm auf dem Rechner des Clients zu starten, das Zugriff auf die Aktivität des Benutzers hat und dauerhaft ausgeführt werden kann, muss dies nicht über ein wiederholt gestartetes Applet geschehen. Da der Demonstrator speziell auf ein „Nachrüsten" einer bestehenden Web-Applikation ausgerichtet ist, ist es in diesem Kontext nicht möglich ein Programm außerhalb des Browsers laufen zu lassen.

Identity Management Die Authentifikation im Demonstrator kann über eine Kombination aus Wissen und Besitz erfolgen (Siehe Kapitel 5.3 und 8.9). Als Wissen dient der Vorweis eines geheimen Passwort und als Besitz der zum Benutzer gehörende WIBU-KEY. Ursprünglich sollte der Nachfolger des WIBU-KEY, Codemeter, von Wibu Systems integriert werden, allerdings gab es für diesen noch keine Java API. Die Schwächen des WIBU-KEY werden in Kapitel 5.3 erläutert.

Da es kein einheitliches Framework für die Integration eines Hardwaretoken in eine Webapplikation gibt bzw. sehr wenig Literatur zu dieser Thematik, erfolgt die Authentifikation durch den WIBU-KEY durch ein von den Autoren entworfenen Challenge-Response-Verfahren. Eine genaue Überprüfung dieses Challenge-Response-Verfahren wurde nicht durchgeführt, da nur gezeigt werden sollte, dass es prinzipiell möglich ist ein Authentifikationsverfahren mit Token in eine Web-Applikation nachträglich zu integrieren. Daher wird von der Übernahme, ohne eine genaue Überprüfung der Verfahrens, in dieser Form abgeraten.

Die von SUN gegründete Liberty Alliance hat mit ihrer ersten Spezifikation ID-FF einen soliden Grundbaustein für Federated Identity geleistet. Das Konkurrenzprodukt von Microsoft WS-* hat zuletzt einen global Player an Liberty verloren. Aufgrund von Kundennachfragen hat IBM nun auch in ihren Tivoli Access Manager das Liberty Protokoll implementiert. Ausserdem ist IBM ein Board

Member bei Liberty geworden [Gon04]. Dies verleiht der Bedeutung der Liberty Alliance ein zusätzliches Gewicht, da damit ein weiterer wichtiger Industrieanbieter beigetreten ist. Es gibt eine große Nachfrage der Industrie an Systemen zur Identity Federation. In immer mehr Produkten werden Konzepte zur Identity Federation integriert.

Die API von SourceID.Java hat die Implementierung des Liberty Protokolls in eine Applikation erleichtert. Allerdings erforderte die Integration eine sehr große Einarbeitungszeit und ein genaues Verständnis des Liberty Protokolls. Mit SourceID.Java ist es nicht auf die Schnelle möglich, eine Web-Applikation um Identity Federations zu erweitern. Die Tatsache, dass es sehr wenig Literatur und Austauschmöglichkeiten mit anderen Entwicklern gibt, erschwerte die Arbeit mit SourceID.Java wesentlich. Es wäre wünschenswert, wenn in Zukunft das Open-Source-Project SourceID.Java mehr den Open-Source-Gedanken aufnimmt und auch andere Entwickler an der Arbeit an SourceID mit einbezieht. Es ist zu hoffen, das hier das neue Project „SourceID Liberty”, welches einen eigenständigen Server darstellt, in diesem Punkt besser durchdacht ist.

Ausblick Es ergaben sich während der Arbeit an dem Demonstrator weitere Ideen, die keinen Eingang in die Implementation gefunden haben. Folgende Auflistung gibt einen Überblick:

Audit Ein Audit stellt die Protokollierung aller Handlungen dar, die ein Mitarbeiter mit der Software ausführen kann. Eine solches Mitschreiben in eine Datenbank kann leicht als ein Aspect in AspectJ formuliert werden. Es wurde davon Abstand genommen, da die rechtliche Betrachtung dieses Themenkomplexes außerhalb des Fokus dieser Diplomarbeit liegt. Technisch gesehen ist dieses Logbuch lediglich eine Tabelle in der Datenbank, in der zu dem Benutzernamen die Eingaben mit den entsprechenden Klassen gespeichert werden. Komplexer ist es, auf diesem Logbuch Auswertungen über das Benutzerverhalten durchzuführen. Insbesonere könnte man Auswertungen über die zeitliche Dauer eines Workflows sowie die bearbeiteten Workflows eines Benutzers anstellen. Zusätzlich könnte man aus den Aufzeichnungen Auswertungen über die ausgeführten Workflows machen um somit die darunterliegende abgesicherte Software zu optimieren.

Chinese Wall Ein Benutzer kann beliebig häufig an dem System angemeldet sein und auch unterschiedlichste Aufgaben ausführen. Chinese Wall bedeutet hier eine dynamische Begrenzung dieser Freiheit. In einer komplexeren Beschreibungssprache könnten hierfür Anforderungen eingefügt werden, so dass ein Mitarbeiter sich z.B. nur einmal anmelden oder immer nur Vorgänge einer Firma bearbeiten darf.

Validierung der Methodenaufrufe Der SecurityAspect dieser Diplomarbeit stellt in seiner jetzigen Form ein Schutz in der Breite dar. Es wird jeweils

die ActionClass des Struts-Framework gekapselt, die die weiteren Aktionen auf die Eingabe auslöst. Ein ergänzendes Modell könnte hier ansetzen und eine Kontrolle in der Tiefe durchführen. Hier gibt es verschiedene Einsatzmöglichkeiten. Es kann kontrolliert werden, inwieweit Methoden zur Ausführung kommen, die so sonst nicht ausgeführt werden. Auf diesem Weg könnten sog. Eastereggs in einer Software erkannt werden. Es könnte auch untersucht werden, inwieweit sich die in [BGP04] vorgestellten Wrapper aufspüren lassen. Es könnte also u.U. eine Aussage über die Sicherheit der JRE getroffen werden.
Daneben wäre die Möglichkeit des Caching gegeben. Hiermit kann der für den Benutzer entscheidende Seitenaufbau beschleunigt werden, falls die Daten der Webseite zu Teilen eine statische Antwort auf seine Eingaben ist.

Strutsbasierung Als Basis des SecurityAspect haben wir das Struts-Framework verwendet. Es wird als Pointcut die ActionClass des Frameworks verwendet. Um den Aspect für beliebige Javaprogramme verwenden zu können ist eine Veränderung des Pointcut nötig und einige weitere Anpassungen im Sourcecode des Aspects. Struts wurde gewählt, da hier die Besonderheiten der Webkommunikation versteckt werden und eine einheitliche Schnittstelle existiert. Der Schritt vom Framework weg bedeutet, dass ab dann statt dieser einen Methode des Frameworks drei Methoden gekapselt werden müssen. Innerhalb des Aspects wurde keine Funktionalität von Struts verwendet, so dass hier keine Probleme bei einer Portierung entstehen sollten.

Verifikation von Workflows Ein Workflow wird von einem Administrator toolgestützt erzeugt. In der momentanen Version hat der Administrator keine Möglichkeit die Korrektheit des von ihm erzeugten Workflows zu kontrollieren, da es kein formales Modell gibt. Ein einfacher Ansatz, dem Administrator eine Möglichkeit zu geben, die Auswirkungen von Änderungen an den Workflows zu testen, ist eine Sammlung von positiven und negativen Beispielen. Einem Workflow könnte eine Testsuite zugeordnet werden, die nach einer Änderung getestet wird. Darüber hinaus kann durch das Tool verlangt werden, dass bevor eine Änderung zugelassen wird, erst ein Testfall angegeben werden muss, für den die Änderung erforderlich ist. Dies würde dem Testfirst-Paradigma des „Extreme Programming" (siehe [NMMB04, SS01]) entsprechen.

Auf der Basis der durch die Suite erzeugten Workflows ist es denkbar mit formalen Methoden die Einhaltung von Bedingungen zu kontrollieren. Hier ist es denkbar Exklusivitätsbedingungen zu überprüfen.

Workflowerzeugung Das Web-Tool aus der Trainingssuite kann lediglich serielle Workflows erzeugen. Die verwendete Definition erlaubt allerdings auch Verzweigungen und Schleifen als Sprachelemente. Die Limitierung in der

Erzeugung der Workflows hat wie in 8.1 beschrieben seine Ursache im Zustand der Applikation nach einem Durchlauf. Eine mögliche Lösung diesem Problem zu begegnen besteht in der Erzeugung und Speicherung des Zustand der Applikation zu jedem Zustand des Workflows (vgl. 8.1)

Implementierung des Web Tools In der Implementierung des Web Tools werden intern Speicherstrukturen verwendet die besser in Form von XML-Strukturen formulierbar wären. Auch hat sich gezeigt, dass der Entwurf der graphischen Oberfläche sich als wesentlich schwerer dargestellt hat als erwartet.

Verwendung von Standards In einer Weiterentwicklung des Konzepts gilt es stärker bestehende Standards zur Modellierung und Beschreibung der Sicherheitsarchitektur heranzuziehen. Insbesondere sind hier zu SAML (vgl. [HM04]) und SUNs WBEM-Services (vgl. [Prob]) zu nennen.

HTML Verifizierung Durch das im HTMLRewriter vorgestellte Konzept können die erstellten Webseiten auf Korrektheit untersucht werden. Es wäre denkbar verschiedene Angriffe, wie sie im Kapitel 6.3 vorgestellt werden, auf diesem Weg erkennen zu können und dynamisch auf diese zu reagieren. Es könnten beispielsweise alle Javascriptanweisungen aus einer Seite herausgefiltert werden oder Modelle für Cross-Site Script Angriffe zur Anwendung kommen. Hierfür wird ein spezieller HTML-Parser benötigt.

Der Demonstrator verlässt sich auf eine Wohlgeformtheit der erzeugten Seite und beschränkt sich auf eine simple Ersetzungsstrategie. Eine allgemeinere Anwendbarkeit des bestehenden Aspects benötigt ebenfalls einen speziellen HTML-Parser. Am Beispiel des JPetStore ist zu sehen, das ein Browser den schliessenden Body-Tag nicht unbedingt benötigt. Die Ersetzungsstrategie aber benötigt den HTML-Tag. Um den Aspect allgemein verwenden zu können, müssen solche Probleme beachtet werden.

Weiterreichende proaktive Maßnahmen Die Proaktivität wurde in dem Demonstrator mit Hilfe eines Applets imtegriert, das in Kapitel 7.1 beschrieben ist. Das Applet registriert eine Aktivität nur, wenn der Benutzer zwischen Webseiten wechselt. Eine weitreichendere Überwachung des Benutzers, z.B. die Registrierung von Tastatureingaben oder Mausbewegungen, ist denkbar.

A Erklärung zur Diplomarbeit

Erklärung zur Diplomarbeit gemäss §19 Abs. 6 DPO/AT

Hiermit versichern wir, dass die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt wurde. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Nicolai Kuntze
Darmstadt, den 17. Februar 2005

Thomas Rauch
Darmstadt, den 17. Februar 2005

B Abbildungsverzeichnis

Abbildungsverzeichnis

C Listingverzeichnis

Listings

D Abkürzungsverzeichnis

Abkürzungsverzeichnis

AAA .... Authentication, Autorisation, Accounting
ACDB .... Access Control Database
ACID .... Atomicity, Consistency, Isolation, Durability
ajc .... AspectJ Compiler
AOP .... Aspect-Oriented Programming
API .... Application Programming Interface
ASP .... Application Service Provider
BMWA .... Bundesministerium fur Wirtschaft und Arbeit
BSD .... Berkeley Software Design
BSI .... Bundesamt fur Sicherheit in der Informationstechnik
CA .... Certification Authority
Cobit .... Control Objectives for Information and related Technology
DES .... Data Encryption Standard
DOM .... Document Object Model
FEAL .... Fast Data Encipherment Algorithm
FIFO .... First In First Out
FSM .... Finite State Machine
GMITS .... guidelines for the management of IT Security
GSHB .... Grundschutzhandbuch des BSI
HTML .... Hypertext Markup Language
HTTP .... Hypertext Transport Protocol
HTTPS .... Secure Hypertext Transport Protocol
IBM .... International Business Machines Corporation
ID-FF .... Identity Federation Framework
ID-SIS .... Identity Services Interface Specifications
ID-WSF .... Identity Web Services Framework
IDB .... Identity Database
IDP .... Identity Provider
IDS .... Intrusion Detection System
IEC .... International Engineering Consortium (IEC)
IO .... Input/Output
IPRS .... Intellectual Property Rights
ISO .... International Organization for Standardization
IT .... Informationstechnologie
ITSEC .... Information Technology Security Evaluation Citeria
J2EE .... Java 2 Enterprise Edition
JAAS .... Java Authentication and Authorization Service
JAAS .... Java Authorization and Authentication Service

JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment
JSP	Java Server Pages
NIST	National Institute of Standards and Technology
PIN	Personal Identification Number
RAID	Redundant Array of Inexpensive Disks
RBAC	Role Based Access Control
RFC	Internet Requests for Comments
SAML	Security Assertion Markup Language
SAX	Simple API for XML
SOAP	Simple Object Access Protocol
SP	Service Provider
SQL	Structured Query language
SSL	Secure Socket Layer
SSO	Single Sign On
TCSEC	Trusted Computer Systems Evaluation Criteria
TLA	Three Letter Acronyms
TLS	Transport Layer Security
URI	Unified Resource Identifier
W3C	World Wide Web Consortium
WBEM	Web-Based Enterprise Management
XML	extensible markup language
XSS	Cross-site Scripting
ZIO	Zero-Interaction Authentication

E Literaturverzeichnis

Literatur

[97997] ISO/IEC 9798-1. Information technology - Security techniques - Entity authentication - Part 1, 1997.

[AA04] Information Systems Audit und Control Association. Cobit Website, 2004. URL: http://www.isaca.org/.

[AAA+95] G. Alonso, Divyakant Agrawal, Amr El Abbadi, C. Mohan, Roger Gunthor und Mohan Kamath. Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Managemen. In *Proceedings of the IFIP WG8.1 Working Conference on Information Systems Development for Decentralized Organizations. Trondheim, Norway*, 1995. URL: http://citeseer.nj.nec.com/alonso95exoticafmqm.html.

[Abi04] Abigail. Simulating the back button. 2004. URL: http://www.foad.org/~abigail/HTML/Misc/back_button.html.

[ACM01] Vijayalakshmi Atluri, Soon Ae Chun und Pietro Mazzoleni. A Chinese wall security model for decentralized workflow systems. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, Seiten 48–57. ACM Press, 2001. URL: http://doi.acm.org/10.1145/501983.501991.

[AG] Wibukey Systems AG. WIBU-KEY. URL: http://wibu.de/de/wibukey.php.

[AG01] Wibukey Systems AG. Keynote No.2, October 2001. URL: http://www.wibu.com/files/news/kn2.pdf.

[AG03] Wibukey Systems AG. Programmer's Guide WIBU-KEY, 2003. Version 4.00.

[AG04] Wibukey Systems AG. Benutzerhandbuch WIBU-KEY, 2004. Version 4.00.

[And72] J. Anderson. Computer Security Technology Planning Study ESD-TR-73-51. Hanscom AFB, Oct 1972.

[ANSI04] Inc. American National Standards Institute. ANSI INCITS 359-2004 Role Based Access Control. 2004.

[AS87] Shoji Miyaguchi Akihiro Shimizu. Fast Data Encryption Algorithm FEAL. *Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'87*, Seiten 267–278, 1987.

[ASKP00] Gail-Joon Ahn, Ravi Sandhu, Myong Kang und Joon Park. Injecting RBAC to secure a Web-based workflow system. In *Proceedings of the fifth ACM workshop on Role-based access control*, Seiten 1–10. ACM Press, 2000. URL: http://doi.acm.org/10.1145/344287.344295.

[BBF02] Roberto Barbuti, Cinzia Bernardeschi und Nicoletta De Francesco. Checking security of Java bytecode by abstract interpretation. In *Proceedings of the 2002 ACM symposium on Applied computing*, Seiten 229–236. ACM Press, 2002. URL: http://doi.acm.org/10.1145/508791.508839.

[BDLD+03] Siddharth Bajaj, Giovanni Della-Libera, Brendan Dixon, Mike Dusche, Maryann Hondo, Matt Hur, Hal Lockhart, Hiroshi Maruyama, Nataraj Nagaratnam, Andrew Nash, Hemma Prafullchandra und John Shewchuk. Specification: Web Services Federation Language (WS-Federation), 2003. URL: http://www-106.ibm.com/developerworks/webservices/library/ws-fed/.

[BDR86] Bharat Bhargava, John Dilley und John Riedl. RAID: a robust and adaptable distributed system. In *Proceedings of the 2nd workshop on Making distributed systems work*, Seiten 1–7. ACM Press, 1986. URL: http://doi.acm.org/10.1145/503956.503965.

[BGP04] Thomas Bühren, Volker Gruhn und Dirk Peters. Konfigurierbare Sicherheit für Java Laufzeitumgebungen. Seiten 329–340, 2004.

[BHG87] Philip Bernstein, V. Hadzilacos und N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. URL: http://research.microsoft.com/pubs/ccontrol/.

[BLFF] T. Berners-Lee, R. Fielding und H. Frystyk. Hypertext Transfer Protocol - HTTP/1.0. URL: http://citeseer.ist.psu.edu/article/berners-lee96hypertext.html.

[BM03] Amit Basu und Steve Muylle. Authentication in e-commerce. *Commun. ACM*, 46(12):159–166, 2003. URL: http://doi.acm.org/10.1145/953460.953496.

[BN89] D. F. C. Brewer und M. J. Nash. The Chinese Wall Security Policy. In *IEEE Symposium on Security and Privacy*, Seiten 206–214, 1989.

[BS91] E. Biham und A. Shamir. Differential cryptanalysis of FEAL and N-Hash. In D.W. Davies, Hrsg., *Advances in Cryptology — Eurocrypt '91*, Seiten 1–16, Berlin, 1991. Springer-Verlag.

[caf] cafesoft. *Tomcat Security Overview and Analysis*. URL: http://www.cafesoft.com/products/cams/tomcat-security.html.

[CK03] Scott Cantor und John Kemp. Liberty ID-FF Bindings and Profiles Specification. 2003. URL: http://www.projectliberty.org/specs/.

[CN02] Mark D. Corner und Brian D. Noble. Zero-interaction authentication. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, Seiten 1–11. ACM Press, 2002. URL: http://doi.acm.org/10.1145/570645.570647.

[Cor] Netscape Communications Corporation. JavaScript Guide. URL: http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/.

[Cow01] Danny Coward. *Java Servlet Specification*. Sun Microsystems Inc., Version 2.3. Auflage, August 2001.

[DCW+99] Robert Durst, Terrence Champion, Brian Witten, Eric Miller und Luigi Spagnuolo. Testing and evaluating computer intrusion detection systems. *Commun. ACM*, 42(7):53–61, 1999. URL: http://doi.acm.org/10.1145/306549.306571.

[DKM+97] S. Das, K. Kochut, J. Miller, A. Sheth und D. Worah. ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR, 1997. URL: http://citeseer.ist.psu.edu/das96orbwork.html.

[DOT03] *Microsoft: .Net Passport Review Guide*, Juni 2003. URL: http://download.microsoft.com/download/a/f/4/af49b391-086e-4aa2-a84b-ef6d916b2f08/passport_reviewguide.doc.

[Eck01] Claudia Eckert. *IT-Sicherheit*. Oldenbourgh, 2001. ECK c 01:1 1.Ex.

[Eck03] C. Eckert. *IT-Sicherheit*. Oldenbourg Wissenschaftsverlag GmbH, Rosenheimer Strasse 145, D-81671 Mnchen, 2. auflage. Auflage, 2003.

[FE03] Bryan Field-Elliot. *A Developer's Tour*, 2003. URL: http://www.pingidentity.com/downloads/sourceid_dev_tutorial.ppt.

[FK92] D.F. Ferraiolo und D.R. Kuhn. Role Based Access Control. *15th National Computer Security Conference*, 1992. URL: http://csrc.nist.gov/rbac/Role_Based_Access_Control-1992.html.

[Fou] Apache Software Foundation. Security Manager HOW-TO. URL: http://jakarta.apache.org/tomcat/tomcat-5.0-doc/security-manager-howto.html.

[Fou04a] Apache Software Foundation. The Valve Component, 2004. URL: http://jakarta.apache.org/tomcat/tomcat-5.0-doc/config/valve.html.

[Fou04b] The Apache Software Foundation. Jakarta Struts. 2004. URL: http://struts.apache.org/.

[FPHKH00] Batya Friedman, Jr. Peter H. Khan und Daniel C. Howe. Trust online. *Commun. ACM*, 43(12):34–40, 2000. URL: http://doi.acm.org/10.1145/355112.355120.

[Fra97] B. Fraser. RFC 2196 - Site Security Handbook, 1997. URL: http://www.faqs.org/rfcs/rfc2196.html.

[Fri97] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O'REILLY, 1997.

[fSidI92] Bundesamt für Sicherheit in der Informationstechnik. *IT-Sicherheitshandbuch*. Bundesamt für Sicherheit in der Informationstechnik, 1992.

[fSidI97] Bundesamt für Sicherheit in der Informationstechnik. IT-Sicherheitskriterien und Evaluierung nach ITSEC, 1997. URL: http://www.bsi.bund.de/zertifiz/itkrit/itsec.htm.

[fSidI03a] Bundesamt für Sicherheit in der Informationstechnik. BSI-Studien, 2003. URL: http://www.bsi.bund.de/literat/studien/index.htm.

[fSidI03b] Bundesamt für Sicherheit in der Informationstechnik. Grundschutzhandbuch. 2003. URL: http://www.bsi.de/gshb/.

[fSidI04] Bundesamt für Sicherheit in der Informationstechnik. BS7799 Part 1 - Vergleich mit dem IT-Grundschutzhandbuch, 2004. URL: http://www.bsi.bund.de/gshb/deutsch/aktuell/bs7799.htm.

[fTSI96] Institute for Telecommunication Sciences (ITS). FED-STD-1037, Glossary of Telecommunication Terms (1980). 1996. URL: http://www.its.bldrdoc.gov/fs-1037/dir-030/_4486.htm.

[Gon04] Antone Gonsalves. IBM Gets Behind Federated-Identity Standard, 2004. URL: http://www.techweb.com/wire/networking/51000068.

[Gro03] Thomas Groß. Security Analysis of the SAML Single Sign-on Browser/Artifact Profile. 2003. URL: http://www.acsac.org/2003/papers/73.pdf.

[Gro04] ISO 17799 Information Security Group. The ISO 17799 Directory, 2004. URL: http://www.iso-17799.com/index.htm.

[Hai03] Sven Haiges. Tools zum Verstreben. *Java Magazin*, 2003. URL: http://www.javamagazin.de/itr/online_artikel/psecom,id,423,nodeid,11.html.

[Hal94] Neil M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed System Security*, Seiten 151–157, 1994.

[hDG04] The hsqldb Development Group. Hypersonic Database. 2004. URL: http://hsqldb.sourceforge.net/.

[HM01] Pieter H. Hartel und Luc Moreau. Formalizing the safety of Java, the Java virtual machine, and Java card. *ACM Comput. Surv.*, 33(4):517–558, 2001. URL: http://doi.acm.org/10.1145/503112.503115.

[HM04] John Hughes und Eve Maler. Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1. 2004. URL: http://www.oasis-open.org/committees/documents.php?wg_abbrev=security.

[Hor04] Torsten Horn. SSL mit Tomcat. 2004. URL: http://www.torsten-horn.de/techdocs/ssl.htm.

[HSO+04] Michael Herfert, Andreas U. Schmidt, Peter Ochsenschläger, Jürgen Repp, Roland Rieke, Martin Schmucker, Steven Vettermann, Uwe Böttge, Cristina Escaleira und Dirk Rüdiger. Implementierung von Security Policies in offenen Telekollaborationen. In *D-A-CH Security 2004*, Seiten 37–49. P.Horster (Hrsg.), 2004.

[IDS00] Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000. URL: http://doi.acm.org/10.1145/382912.382923.

[Int] Internet2. *Uses of Shibboleth*. URL: http://shibboleth.internet2.edu/shib-uses.html.

[JF03] JasonBrittain und Ian F.Darwin. *Tomcat: The Definitive Guide*. O'Reilly, 1. Auflage, 2003.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier und John Irwin. Aspect-Oriented Programming. 1997. URL: http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-ECOOP97/.

[KR00] David P. Kormann und Aviel D. Rubin. Risks of the Passport Single Signon Protocol. *Computer Networks*, (Vol. 33):51–58, 2000. URL: http://avirubin.com/passport.html.

[KS02] Savith Kandala und Ravi Sandhu. Secure role-based workflow models. In *Proceedings of the fifteenth annual working conference on Database and application security*, Seiten 45–58. Kluwer Academic Publishers, 2002.

[Lab00] RSA Laboratories. RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1, 2000. URL: http://www.rsasecurity.com/rsalabs/node.asp?id=2254.

[LAC03] *Liberty ID-FF Authentication Context Specification*, 11 August 2003. URL: http://www.projectliberty.org/specs/liberty-authentication-context-v1.2.pdf.

[Lad03] Ramnivas Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co., 2003.

[LAP03a] *Liberty Alliance Project White Paper, Liberty Alliance & WS-Federation: A Comparative Overview*, 14 Oktober 2003. URL: https://www.projectliberty.org/resources/whitepapers/wsfed-liberty-overview-10-13-03.pdf.

[LAP03b] Liberty Alliance Project. *Identity Systems and Liberty Specification Version 1.1 Interoperability*, 2003. URL: http://www.projectliberty.org/ resources/whitepapers/Liberty and 3rd Party Identity Systems White Paper.pdf.

[LAP03c] Liberty Alliance Project. Liberty ID-FF Architecture Overview Version 1.2. 2003. URL: http://www.projectliberty.org/specs/liberty-idff-arch-overview-v1.2.pdf.

[Lew03] J. Lewis. Enterprise Identity Management. *Datenschutz und Datensicherheit*, Seiten 533–535, 2003.

[LGK+99] Charlie Lai, Li Gong, Larry Koved, Anthony Nadalin und Roland Schemers. User Authentication and Authorization in the Java Platform. In *15th Annual Computer Security Applications Conference*, Seiten 285–290. IEEE Computer Society Press, 1999. URL: http://java.sun.com/security/jaas/doc/acsac.html.

[Meg04] David Megginson. SAX (Simple API for XML). 2004. URL: http://www.saxproject.org/.

[Meh04] Michael Mehrhoff. Outsourcing unter Sicherheitsaspekten. In *D-A-CH Security 2004*, Seiten 62–70. P.Horster (Hrsg.), 2004.

[Mel05] Ronald Melster. Workflow Systeme, 2005. URL: http://www.software-kompetenz.de/?13155.

[MKL97] Anurag Mendhekar, Gregor Kiczales und John Lamping. RG: A Case-Study for Aspect-Oriented Programming. 1997. URL: http://www2.parc.com/csl/groups/sda/publications/papers/PARC-AOP-RG97/.

[Mos03] Marie-Luise Moschgath. Sicherheitsmanagement (Sommersemester 2003), 2003. URL: http://www.sec.informatik.tu-darmstadt.de /de/lehre/SS03/ sicherheitsmanagement/ index.html.

[MSS88] S. Miyaguchi, A. Shiraisi und A. Shimizu. Fast data encryption algorithm Feal-8. *Review of Electrical Communications Laboratories*, 36(4):433–437, 1988.

[Mur90] Sean Murphy. The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts. *J. Cryptology*, 2(3):145–154, 1990. URL: http://www.isg.rhul.ac.uk/~sean/feal.ps.

[MWW+98] Peter Muth, Dirk Wodtke, Jeanine Weissenfels, Angelika Kotz Dittrich und Gerhard Weikum. From Centralized Workflow Specification to Distributed Workflow Execution. *J. Intell. Inf. Syst.*, 10(2):159–184, 1998. URL: http://dx.doi.org/10.1023/A:1008608810770.

[NMMB04] James Noble, Stuart Marshall, Stephen Marshall und Robert Biddle. Less Extreme Programming. In *CRPIT '30: Proceedings of the sixth conference on Australian computing education*, Seiten 217–226. Australian Computer Society, Inc., 2004.

[OA94] Kazuo Ohta und Kazumaro Aoki. Linear Cryptanalysis of the Fast Data Encipherment Algorithm. *Lecture Notes in Computer Science*, 839:12–16, 1994. URL: http://citeseer.ist.psu.edu/ohta94linear.html.

[Onl05] Heise Online. Microsofts Authentifizierungsdienst Passport nicht mehr bei eBay, 2005. URL: http://www.heise.de/newsticker/result.xhtml?url=/newsticker/meldung/54705.

[PB04] Maja Pusara und Carla E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, Seiten 1–8. ACM Press, 2004. URL: http://doi.acm.org/10.1145/1029208.1029210.

[Proa] Liberty Alliance Project. Liberty Alliance Developer Tutorial. URL: http://www.projectliberty.org/resources/tutorial_draft.pdf.

[Prob] Jeff Prosise. DMTF Web-Based Enterprise Management Initiative. URL: http://www.dmtf.org/standards/wbem.

[Pro85a] High-Availability Linux Project. High Availability, 1985. URL: http://www.linux-ha.org/.

[Pro85b] Jeff Prosise. Foiling Session Hijacking Attempts, 1985. URL: http://msdn.microsoft.com/msdnmag/issues/04/08/WickedCode/.

[Pro04] Jakarta Tomcat Project. SSL Configuration HOW-TO. 2004. URL: http://jakarta.apache.org/tomcat/tomcat-4.0-doc/ssl-howto.html.

[PW02] Birgit Pfitzmann und Michael Waidner. Privacy in browser-based attribute exchange. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, Seiten 52–62. ACM Press, 2002. URL: http://doi.acm.org/10.1145/644527.644533.

[Ram98] Raghu Ramakrishnan. *Database management systems*. McGraw-Hill, Inc., 1998.

[Rob04] Paul Roberts. Vendors team on WS-Federation standard, 2004. URL: http://www.infoworld.com/article/04/05/25/HNwsfed_1.html.

[RW03a] J. Rouault und T. Wason. Liberty Bindings and Profiles Specification. 2003. URL: http://www.projectliberty.org/specs/.

[R.W03b] R.Wright. 2003 CSI/FBI computer security survey, 2003. URL: http://www.security.fsu.edu/docs/FBI2003.

[RWL+00] C. Rigney, S. Willens, Livingston, A. Rubens, Merit, W. Simpson und Daydreamer. RFC 2865 - Remote Authentication Dial In User Service RADIUS, 2000. URL: http://www.faqs.org/rfcs/rfc2865.html.

[San98] R. Sandhu. Role-based access control. In *Advances in Computers*, Jgg. 46, Seiten 237–286. M. Zelkowitz Eds. Academic, 1998.

[SBW01] Anjana Susarla, Anitesh Barua und Andrew B. Whinston. Myths about Outsourcing to Application Service Providers. 2001.

[SCFY96] R. Sandhu, E.J. Coyne, H.L. Feinstein und C.E. Youman. Rolebased access control models. *IEEE Computer Volume 29*, 1996.

[Sch03] Fred B. Schneider. Least Privilege and More. *j-IEEE-SEC-PRIV*, 1(5):55–59, September/ Oktober 2003. URL: http://csdl.computer.org/comp/mags/sp/2003/05/j5055abs.htm http://csdl.computer.org/dl/mags/sp/2003/05/j5055.pdf.

[Sch04] Marc Schönefeld. Seitenkanalangriffe auf die Sicherheit von Javabasierten Softwarearchitekturen. In *D.A.CH Security 2004*, Seiten 341–348. P. Horster, 2004.

[SLZ+04] O. Slattery, R. Lu, J. Zheng, F. Byers und X. Tang. Stability Comparison of Recordable Optical Discs?A Study of Error Rates in Harsh Conditions, 2004. URL: http://www.nist.gov/public_affairs/techbeat/tb2004_1208.htm#cddvd.

[SNS88] Jennifer G. Steiner, B. Clifford Neuman und Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *USENIX Winter*, Seiten 191–202, 1988.

[SO04] Antti Salovaara und Antti Oulasvirta. Six modes of proactive resource management: a user-centric typology for proactive behaviors. In *Proceedings of the third Nordic conference on Human-computer interaction*, Seiten 57–60. ACM Press, 2004. URL: http://doi.acm.org/10.1145/1028014.1028022.

[sou03] *SourceID Licensing*, 2003. URL: http://www.pingidentity.com/license.

[SPP+04] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu und Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, Seiten 298–307. ACM Press, 2004. URL: http://doi.acm.org/10.1145/1030083.1030124.

[SS75] JEROME H. SALTZER und MICHAEL D. SCHROEDER. The Protection of Information in Computer Systems. 1975. URL: http://web.mit.edu/Saltzer/www/publications/protection/.

[SS96] Ravi Sandhu und Pierangela Samarati. Authentication, access control, and audit. *ACM Comput. Surv.*, 28(1):241–243, 1996. URL: http://doi.acm.org/10.1145/234313.234412.

[SS01] Suzanne Smith und Sara Stoecklin. What we can learn from extreme programming. *J. Comput. Small Coll.*, 17(2):144–151, 2001.

[SSL04] sslext - The Struts SSL Extension for HTTP/HTTPS switching. 2004. URL: http://sslext.sourceforge.net/.

[ST85] Pagadala J. Suresh und Palaniyappan Thiagarajan. JAR files revealed, 1985. URL: http://www-106.ibm.com/developerworks/java/library/j-jar/.

[STA85] DEPARTMENT OF DEFENSE STANDARD. TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA, 1985. URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html.

[SUN85] SUN. jarsigner - JAR Signing and Verification Tool, 1985. URL: http://java.sun.com/j2se/1.3/docs/tooldocs/win32/jarsigner.html.

[Sun04] Sun. Java Authentication and Authorization Service (JAAS) Overview, 2004. URL: http://java.sun.com/products/jaas/overview.html.

[SZ03] Pawel Slowikowski und Krzysztof Zielinski. Comparison Study of Aspect-oriented and Container Managed Security. *AAOS2003: Analysis of Aspect Oriented Software*, 2003.

[TC03] Peter Thompson und Darryl Champagne. Liberty ID-FF Implementation Guidlines. 2003. URL: http://www.projectliberty.org/specs/.

[Ten00] David Tennenhouse. Proactive computing. *Commun. ACM*, 43(5):43–50, 2000. URL: http://doi.acm.org/10.1145/332833.332837.

[Tri04] Inc. Tripwire. Tripwire, 2004. URL: http://www.tripwire.org/.

[Vas04] Alexandre Vasseur. *Aspectwerkz Website*, 2004. URL: http://aspectwerkz.codehaus.org/.

[VCF+00] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege und D. Spence. RFC 2904 - AAA Authorization Framework, 2000. URL: http://www.faqs.org/rfcs/rfc2904.html.

[Voe00] Markus Voelter. Aspectj-Oriented Programming in Java. 2000. URL: http://www.voelter.de/data/articles/aop/aop.html.

[Vog04] Marko Vogel. Identity management Kosten und Nutzen. *D.A.CH Security*, 2004.

[VVK03] Giovanni Vigna, Fredrik Valeur und Richard A. Kemmerer. Designing and implementing a family of intrusion detection systems. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, Seiten 88–97. ACM Press, 2003. URL: http://doi.acm.org/10.1145/940071.940084.

[W3C00] W3C. *Simple Object Access Protocol (SOAP) 1.1*, 2000. URL: http://www.w3c.org/TR/SOAP.

[w3c04] w3c. XML Schema. 2004. URL: http://www.w3.org/XML/Schema.

[Wik] the free encyclopedia Wikipedia. Three-tier (computing). URL: http://en.wikipedia.org/wiki/Three-tier_(computing).

[WJP03] Bart De Win, Wouter Joosen und Frank Piessens. AOSD & Security: a practical assessment. February 28, 2003.

[Woe04] Thomas Woelfer. Cross-Site Scripting und SQL-Injection, 2004. URL: http://www.tecchannel.de/internet/1423/1.html.